- Jargon-free Tips & Advice
- Step-by-step Tutorials
- Clear Full Colour Guides

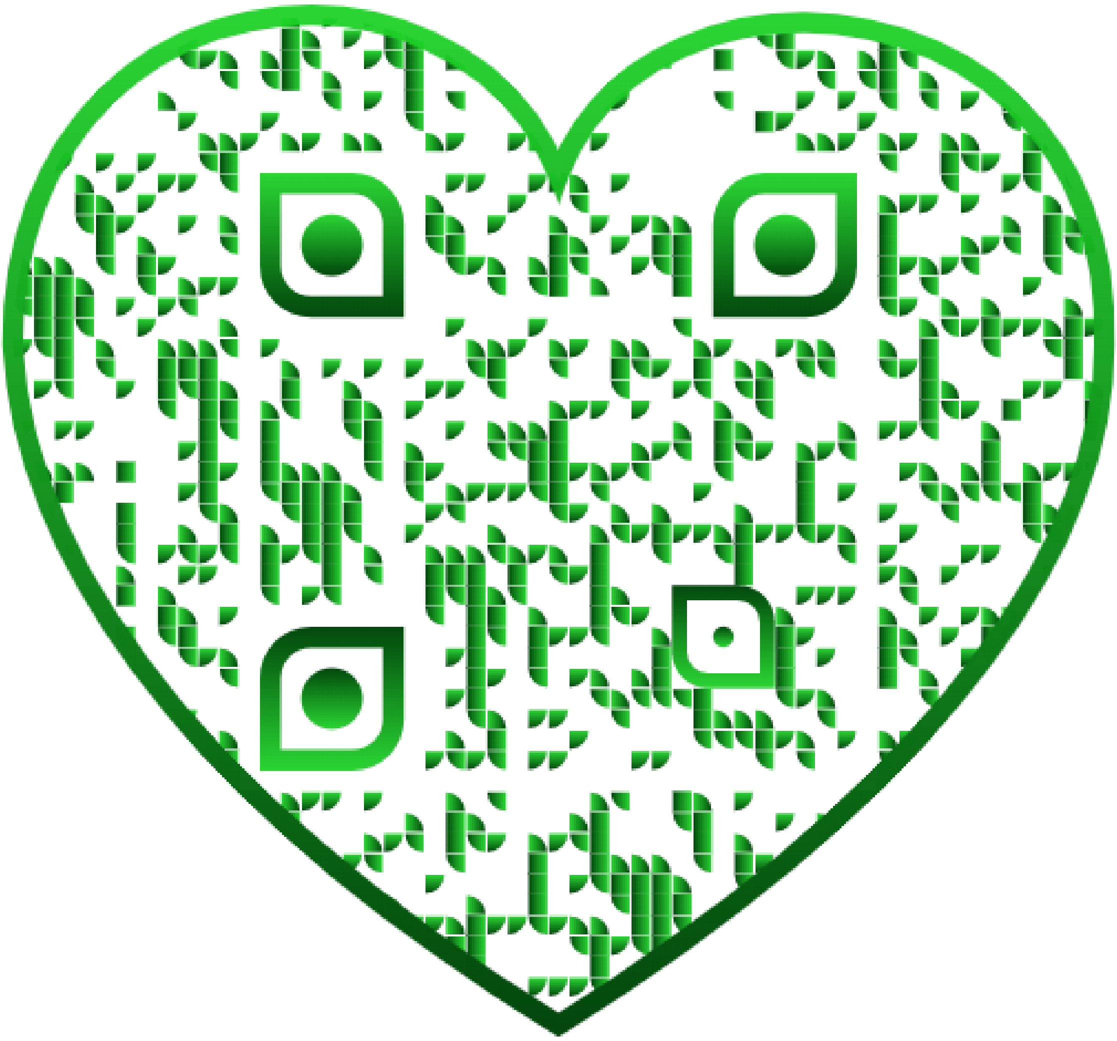
C++ & Python For Beginners

Get started with new coding projects

Over 455 Tips & Hints inside

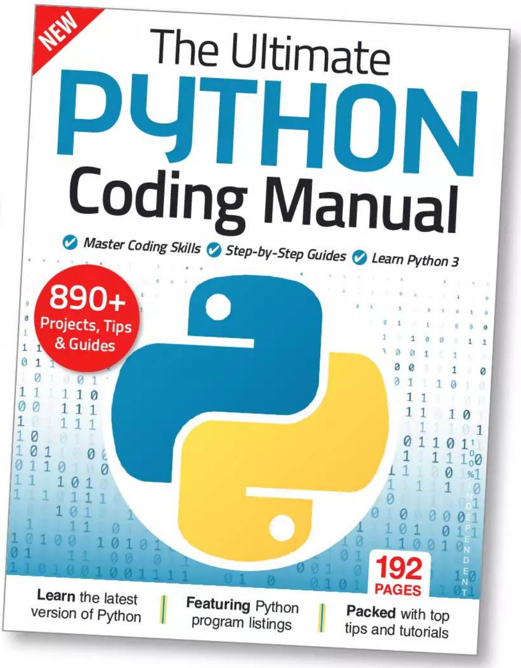
100% INDEPENDENT

Papercut



Get Your Exclusive FREE Gift Worth £9.99 Here!

Download Your FREE Copy of *The Ultimate Python Coding Manual*


NEW
The Ultimate
PYTHON
Coding Manual
Master Coding Skills
Step-by-Step Guides
Learn Python 3
890+ Projects, Tips & Guides
192 PAGES
Learn the latest version of Python
Featuring Python program listings
Packed with top tips and tutorials

Head over to your web browser and follow these simple instructions...

1/ Enter the following URL: **www.pclpublications.com/exclusives**

2/ Sign up/in and from the listings of our exclusive customer downloads, highlight the *The Ultimate Python Coding Manual* option.

3/ Enter your unique download code (Listed below) in the "Enter password to download" bar.

4/ Click the Download Now! Button and your digital manual will automatically download.

5/ Your file is a high resolution PDF file, which is compatible with the majority of customer devices/platforms.

Exclusive Download Code: `PCL64792CM`

NOTE: This is a limited reader offer running from 1st June 2024 to 1st September 2024.
Terms and conditions apply and can be found at www.pclpublications.com

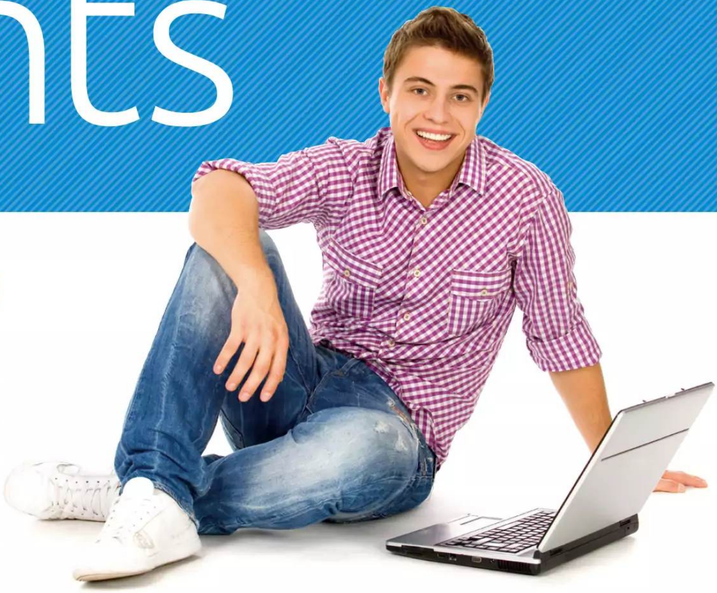
C++ & Python For Beginners

Coding with C++ & Python For Beginners is the first and only choice if you are new adopter and want to learn everything you'll need to get started with coding and programming. This independent guide is crammed with helpful guides and step-by-step fully illustrated tutorials, written in plain easy to follow English. Over the pages of this new user guide you will clearly learn all you need to know about coding your own amazing apps. With this unofficial instruction manual at your side no problem will be unsolvable, no question unanswered as you learn, explore and enhance your programming skills.

Papercut

www.pclpublications.com

Contents



6 Say Hello to Python

8 Why Python?
10 Equipment You Will Need
12 Getting to Know Python
14 How to Set Up Python in Windows
16 How to Set Up Python on a Mac
18 How to Set Up Python in Linux

20 Getting Started with Python

22 Starting Python for the First Time
24 Your First Code
26 Saving and Executing Your Code
28 Executing Code from the Command Line
30 Numbers and Expressions
32 Using Comments
34 Working with Variables
36 User Input
38 Creating Functions
40 Conditions and Loops
42 Python Modules

44 Say Hello to C++

46 Why C++?
48 Equipment Needed
50 How to Set Up C++ in Windows
52 How to Set Up C++ on a Mac
54 How to Set Up C++ in Linux
56 Other C++ IDEs to Install

58 C++ Fundamentals

60 Your First C++ Program
62 Structure of a C++ Program
64 Compile and Execute
66 Using Comments
68 Variables
70 Data Types
72 Strings
74 C++ Maths

FREE CODE DOWNLOAD!

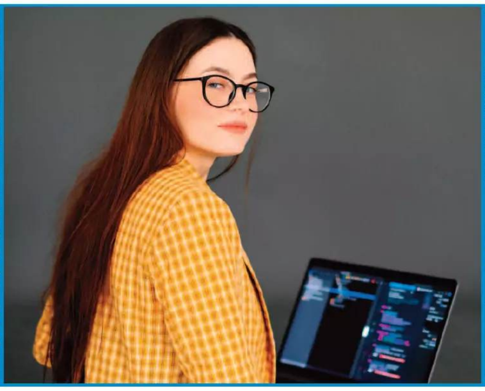
50 Complete Programs

Over 20,000 lines of code!

Visit: www.pclpublications.com/exclusives

Please note: Sign up is required to download

"…technology is everywhere and it's all connected through programming. Your TV, microwave, in-car entertainment and the Internet itself are all reliant on good programming to make them work the way you want them to…"



Say Hello to Python

There are many different programming languages available to learn and use. Some are complex and incredibly powerful and some are extremely basic and used as minor utilities for operating systems. Python sits somewhere in the middle, combining ease of use with a generous helping of power that allows the user to create minor utilities, a range of excellent games and performance-heavy computational tasks.

However, there's more to Python than simply being another programming language. It has a vibrant and lively community behind it that shares knowledge, code and project ideas; as well as bug fixes for future releases. It's thanks to this community that the language has grown and thrived and now it's your turn to take the plunge and learn how to program in Python.

The first half of this book helps you get started with the latest version of Python and from there guide you on how to use some of the most common and interesting functions and features of the language. Before long, you will be able to code your own helpful system tools, text adventures and even control a character as they move around the screen.

Why Python?

There are many different programming languages available for the modern computer, and some still available for older 8 and 16-bit computers too. Some of these languages are designed for scientific work, others for mobile platforms and such. So why choose Python out of all the rest?

PYTHON POWER

Ever since the earliest home computers were available, enthusiasts, users and professionals have toiled away until the wee hours, slaving over an overheating heap of circuitry to create something akin to magic.

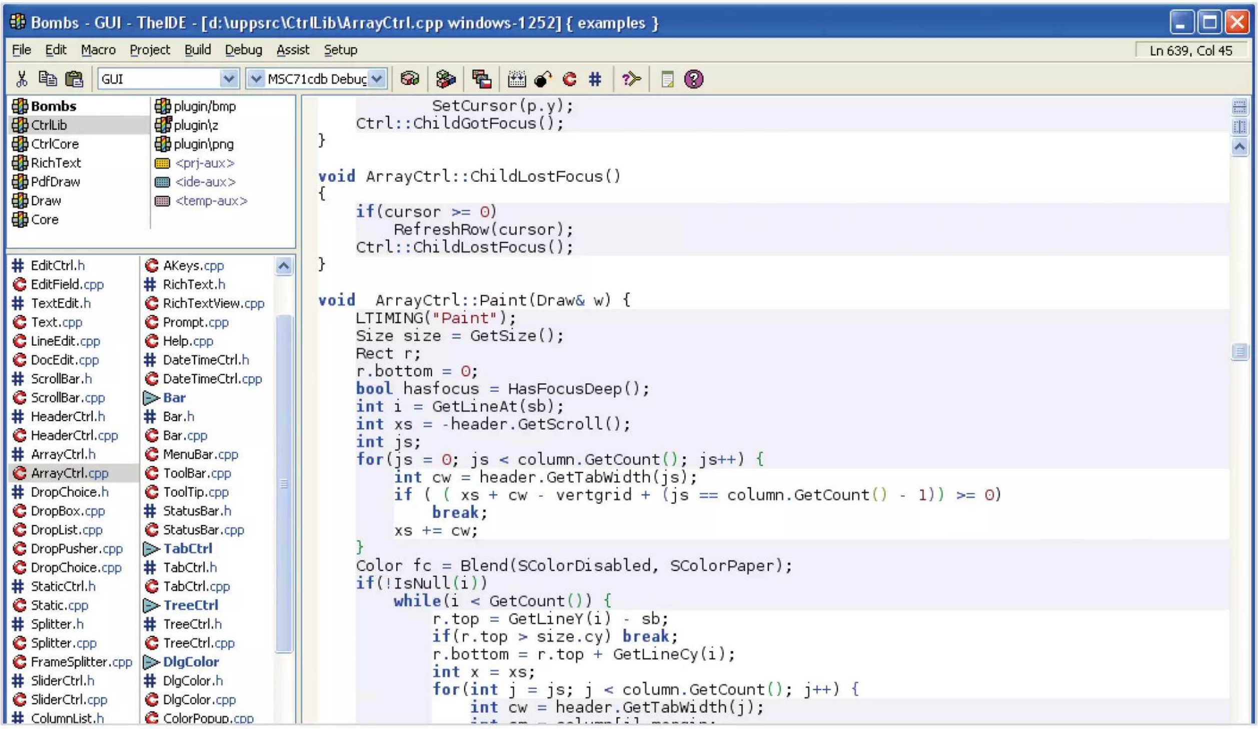
These pioneers of programming carved their way into a new frontier, forging small routines that enabled the letter 'A' to scroll across the screen. It may not sound terribly exciting to a generation that's used to ultra-high definition graphics and open world, multi-player online gaming. However, forty-something years ago it was blindingly brilliant.

Naturally these bedroom coders helped form the foundations for every piece of digital technology we use today. Some went on to become chief developers for top software companies, whereas others pushed the available hardware to its limits and founded the billion pound gaming empire that continually amazes us.

Regardless of whether you use an Android device, iOS device, PC, Mac, Linux, Smart TV, games console, MP3 player, GPS device built-in to a car, set-top box or a thousand other connected and 'smart' appliances, behind them all is programming.

All those aforementioned digital devices need instructions to tell them what to do, and allow them to be interacted with. These instructions form the programming core of the device and that core can be built using a variety of programming languages.

The languages in use today differ depending on the situation, the platform, the device's use and how the device will interact with its


Bombs - GUI - TheIDE - [d:\uppsrc\CtrlLib\ArrayCtrl.cpp windows-1252] { examples }
File Edit Macro Project Build Debug Assist Setup
Ln 639, Col 45
GUI
MSC71cdb Debug
Bombs
CtrlLib
CtrlCore
RichText
PdfDraw
Draw
Core
plugin/bmp
plugin\z
plugin\png
<prj-aux>
<ide-aux>
<temp-aux>
EditCtrl.h
EditField.cpp
TextEdit.h
Text.cpp
LineEdit.cpp
DocEdit.cpp
ScrollBar.h
ScrollBar.cpp
HeaderCtrl.h
HeaderCtrl.cpp
ArrayCtrl.h
ArrayCtrl.cpp
DropChoice.h
DropBox.cpp
DropList.cpp
DropPusher.cpp
DropChoice.cpp
StaticCtrl.h
Static.cpp
Splitter.h
Splitter.cpp
FrameSplitter.cpp
SliderCtrl.h
SliderCtrl.cpp
ColumnList.h
AKeys.cpp
RichText.h
RichTextView.cpp
Prompt.cpp
Help.cpp
DateTimeCtrl.h
DateTimeCtrl.cpp
Bar
Bar.h
Bar.cpp
MenuBar.cpp
ToolBar.cpp
ToolTip.cpp
StatusBar.h
StatusBar.cpp
TabCtrl
TabCtrl.h
TabCtrl.cpp
TreeCtrl
TreeCtrl.h
TreeCtrl.cpp
DlgColor
DlgColor.h
DlgColor.cpp
ColorPopup.cpp

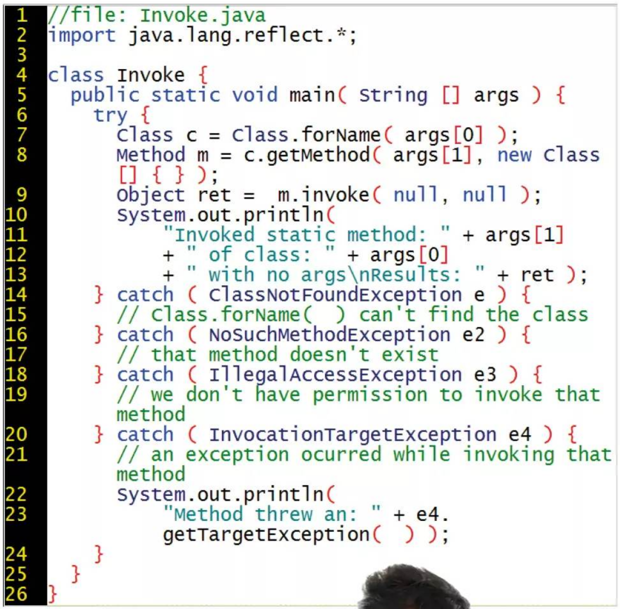
```
            SetCursor(p.y);
        Ctrl::ChildGotFocus();
}

void ArrayCtrl::ChildLostFocus()
{
        if(cursor >= 0)
            RefreshRow(cursor);
        Ctrl::ChildLostFocus();
}

void  ArrayCtrl::Paint(Draw& w) {
        LTIMING("Paint");
        Size size = GetSize();
        Rect r;
        r.bottom = 0;
        bool hasfocus = HasFocusDeep();
        int i = GetLineAt(sb);
        int xs = -header.GetScroll();
        int js;
        for(js = 0; js < column.GetCount(); js++) {
            int cw = header.GetTabWidth(js);
            if ( ( xs + cw - vertgrid + (js == column.GetCount() - 1)) >= 0)
                break;
            xs += cw;
        }
        Color fc = Blend(SColorDisabled, SColorPaper);
        if(!IsNull(i))
            while(i < GetCount()) {
                r.top = GetLineY(i) - sb;
                if(r.top > size.cy) break;
                r.bottom = r.top + GetLineCy(i);
                int x = xs;
                for(int j = js; j < column.GetCount(); j++) {
                    int cw = header.GetTabWidth(j);
```

C++ is usually reserved for more complex programs, operating systems, games and so on.

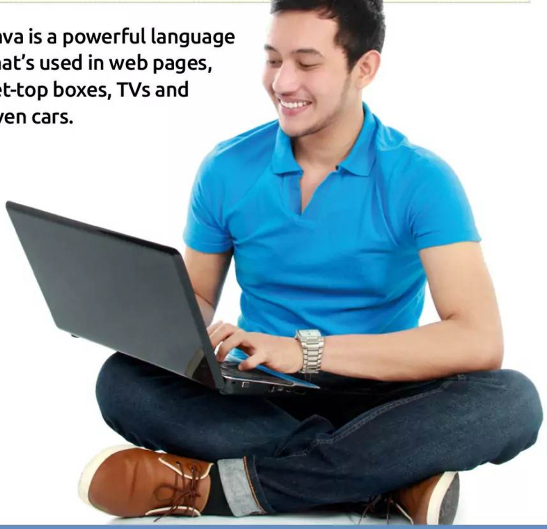
environment or users. Operating systems, such as Windows, macOS and such are usually a combination of C++, C#, assembly and some form of visual-based language. Games generally use C++ whilst web pages can use a plethora of available languages such as HTML, Java, Python and so on.

More general-purpose programming is used to create programs, apps, software or whatever else you want to call them. They're widely used across all hardware platforms and suit virtually every conceivable application. Some operate faster than others and some are easier to learn and use than others. Python is one such general-purpose language.

Python is what's known as a High-Level Language, in that it 'talks' to the hardware and operating system using a variety of arrays, variables, objects, arithmetic, subroutines, loops and countless more interactions. Whilst it's not as streamlined as a Low-Level Language, which can deal directly with memory addresses, call stacks and registers, its benefit is that it's universally accessible and easy to learn.



```
//file: Invoke.java
import java.lang.reflect.*;

class Invoke {
  public static void main( String [] args ) {
    try {
      Class c = Class.forName( args[0] );
      Method m = c.getMethod( args[1], new Class
      [] { } );
      Object ret =  m.invoke( null, null );
      System.out.println(
          "Invoked static method: " + args[1]
          + " of class: " + args[0]
          + " with no args\nResults: " + ret );
    } catch ( ClassNotFoundException e ) {
      // Class.forName(  ) can't find the class
    } catch ( NoSuchMethodException e2 ) {
      // that method doesn't exist
    } catch ( IllegalAccessException e3 ) {
      // we don't have permission to invoke that
      method
    } catch ( InvocationTargetException e4 ) {
      // an exception ocurred while invoking that
      method
      System.out.println(
          "Method threw an: " + e4.
          getTargetException(  ) );
    }
  }
}
```

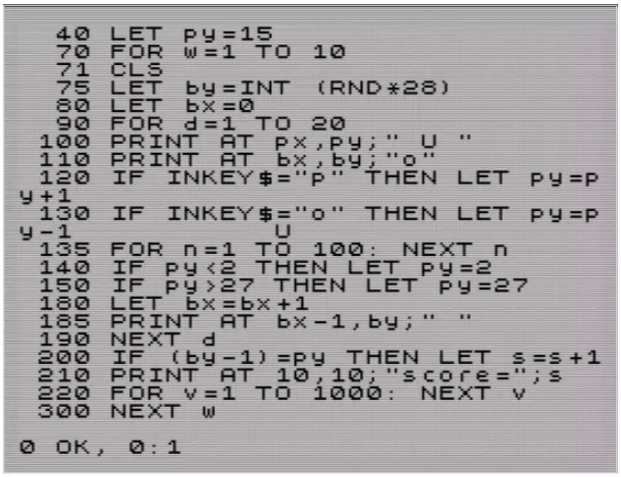


Java is a powerful language that's used in web pages, set-top boxes, TVs and even cars.

Python was created over twenty six years ago and has evolved to become an ideal beginner's language for learning how to program a computer. It's perfect for the hobbyist, enthusiast, student, teacher and those who simply need to create their own unique interaction between either themselves or a piece of external hardware and the computer itself.

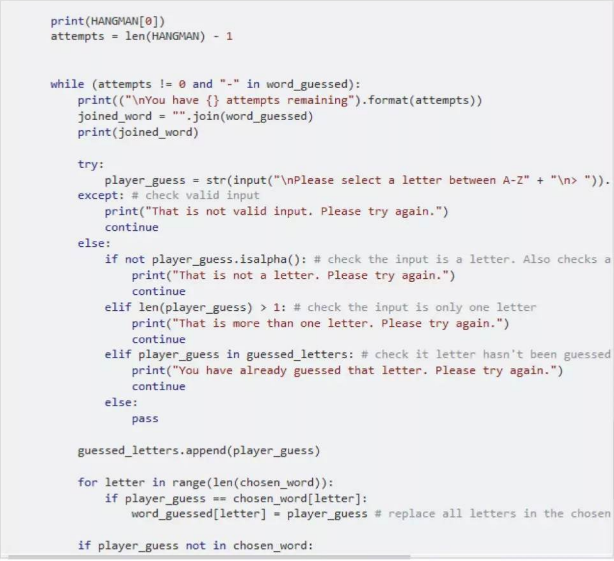
Python is free to download, install and use and is available for Linux, Windows, macOS, MS-DOS, OS/2, BeOS, IBM i-series machines, and even RISC OS. It has been voted one of the top five programming languages in the world and is continually evolving ahead of the hardware and Internet development curve.

So to answer the question: why python? Simply put, it's free, easy to learn, exceptionally powerful, universally accepted, effective and a superb learning and educational tool.



```
  40 LET py=15
  70 FOR w=1 TO 10
  71 CLS
  75 LET by=INT (RND*28)
  80 LET bx=0
  90 FOR d=1 TO 20
 100 PRINT AT px,py;" U "
 110 PRINT AT bx,by;"o"
 120 IF INKEY$="p" THEN LET py=p
y+1
 130 IF INKEY$="o" THEN LET py=p
y-1          U
 135 FOR n=1 TO 100: NEXT n
 140 IF py<2 THEN LET py=2
 150 IF py>27 THEN LET py=27
 180 LET bx=bx+1
 185 PRINT AT bx-1,by;" "
 190 NEXT d
 200 IF (by-1)=py THEN LET s=s+1
 210 PRINT AT 10,10;"score=";s
 220 FOR v=1 TO 1000: NEXT v
 300 NEXT w

0 OK, 0:1
```

BASIC was once the starter language that early 8-bit home computer users learned.



```
print(HANGMAN[0])
attempts = len(HANGMAN) - 1


while (attempts != 0 and "-" in word_guessed):
    print(("\nYou have {} attempts remaining").format(attempts))
    joined_word = "".join(word_guessed)
    print(joined_word)

    try:
        player_guess = str(input("\nPlease select a letter between A-Z" + "\n> ")).
    except: # check valid input
        print("That is not valid input. Please try again.")
        continue
    else:
        if not player_guess.isalpha(): # check the input is a letter. Also checks a
            print("That is not a letter. Please try again.")
            continue
        elif len(player_guess) > 1: # check the input is only one letter
            print("That is more than one letter. Please try again.")
            continue
        elif player_guess in guessed_letters: # check it letter hasn't been guessed
            print("You have already guessed that letter. Please try again.")
            continue
        else:
            pass

    guessed_letters.append(player_guess)

    for letter in range(len(chosen_word)):
        if player_guess == chosen_word[letter]:
            word_guessed[letter] = player_guess # replace all letters in the chosen

    if player_guess not in chosen_word:
```

Python is a more modern take on BASIC, it's easy to learn and makes for an ideal beginner's programming language.

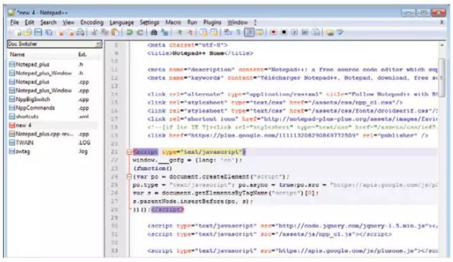
Equipment You Will Need

You can learn Python with very little hardware or initial financial investment. You don't need an incredibly powerful computer and any software that's required is freely available.

WHAT WE'RE USING

Thankfully, Python is a multi-platform programming language available for Windows, macOS, Linux, Raspberry Pi and more. If you have one of those systems, then you can easily start using Python.

Python
PSF
Docs
PyPI
Jobs
Community
python
GO
Socialize
About
Downloads
Documentation
Community
Success Stories
News
Events
Download the latest version for Windows
Looking for a specific release?
Python releases by version number:
Release version
Release date
Click for more


*new 4 - Notepad++
File
Edit
Search
View
Encoding
Language
Settings
Macro
Run
Plugins
Window
?
Doc Switcher
Name
Ext.
Notepad_plus .h
Notepad_plus_Window .h
Notepad_plus .cpp
Notepad_plus_Window .cpp
NppBigSwitch .cpp
NppCommands .cpp
shortcuts .xml
new 4
Notepad_plus.cpp-rev... .cpp
TWAIN .LOG
swtag .log

```
    <meta charset="utf-8">
    <title>Notepad++ Home</title>

    <meta name="description" content="Notepad++: a free source code editor which su
    <meta name="keywords" content="Télécharger Notepad++, Notepad, download, free so

    <link rel="alternate" type="application/rss+xml" title="Follow Notepad++ with RS
    <link rel="stylesheet" type="text/css" href="/assets/css/npp_cl.css"/>
    <link rel="stylesheet" type="text/css" href="/assets/css/fonts/droidserif.css"/
    <link rel="shortcut icon" href="http://notepad-plus-plus.org/assets/images/favi
    <!--[if lte IE 7]><link rel="stylesheet" type="text/css" href="/assets/css/ie67.
    <link href="https://plus.google.com/111113208290869772509" rel="publisher" />

<script type="text/javascript">
window.___gcfg = {lang: 'en'};
(function()
{var po = document.createElement("script");
po.type = "text/javascript"; po.async = true;po.src = "https://apis.google.com/js/pl
var s = document.getElementsByTagName("script")[0];
s.parentNode.insertBefore(po, s);
})();</script>

    <script type="text/javascript" src="http://code.jquery.com/jquery-1.5.min.js"></
    <script type="text/javascript" src="/assets/js/npp_cl.js"></script>

    <script type="text/javascript" src="https://apis.google.com/js/plusone.js"></scr
```

☐ COMPUTER

Obviously you're going to need a computer in order to learn how to program in Python and to test your code. You can use Windows (from XP onward) on either a 32 or 64-bit processor, an Apple Mac or Linux installed PC.

☐ AN IDE

An IDE (Integrated Developer Environment) is used to enter and execute Python code. It enables you to inspect your program code and the values within the code, as well as offering advanced features. There are many different IDEs available, so find the one that works for you and gives the best results.

☐ PYTHON SOFTWARE

macOS and Linux already come with Python preinstalled as part of the operating system, as does the Raspberry Pi. However, you need to ensure that you're running the latest version of Python. Windows users need to download and install Python, which we'll cover shortly.

☐ TEXT EDITOR

Whilst a text editor is an ideal environment to enter code into, it's not an absolute necessity. You can enter and execute code directly from the IDLE but a text editor, such as Sublime Text or Notepad++, offers more advanced features and colour coding when entering code.

☐ INTERNET ACCESS

Python is an ever evolving environment and as such new versions often introduce new concepts or change existing commands and code structure to make it a more efficient language. Having access to the Internet will keep you up-to-date, help you out when you get stuck and give access to Python's immense number of modules.

☐ TIME AND PATIENCE

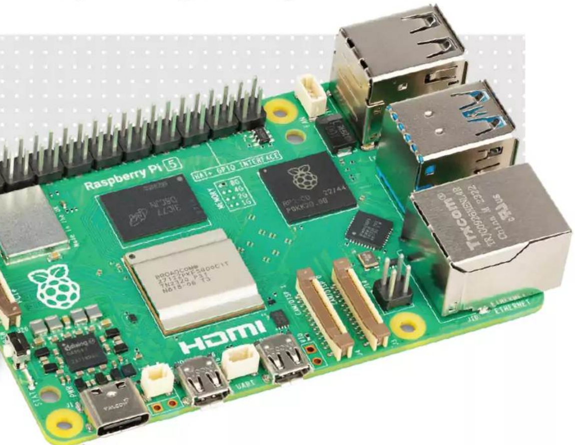
Despite what other books may lead you to believe, you won't become a programmer in 24-hours. Learning to code in Python takes time, and patience. You may become stuck at times and other times the code will flow like water. Understand you're learning something entirely new, and you will get there.

THE RASPBERRY PI

Why use a Raspberry Pi? The Raspberry Pi is a tiny computer that's very cheap to purchase but offers the user a fantastic learning platform. Its main operating system, Raspbian, comes preinstalled with the latest Python along with many Modules and extras.

RASPBERRY PI

The Raspberry Pi 5 is the latest version, incorporating a more powerful CPU, more memory, Wi-Fi and Bluetooth support. You can pick up a Pi 5 for around £59 or as part of a kit depending on the Pi model you're interested in.



FUZE PROJECT

The FUZE is a learning environment built on the latest model of the Raspberry Pi. You can purchase the workstations that come with an electronics kit and even a robot arm for you to build and program. You can find more information on the FUZE at **www.fuze.co.uk**.

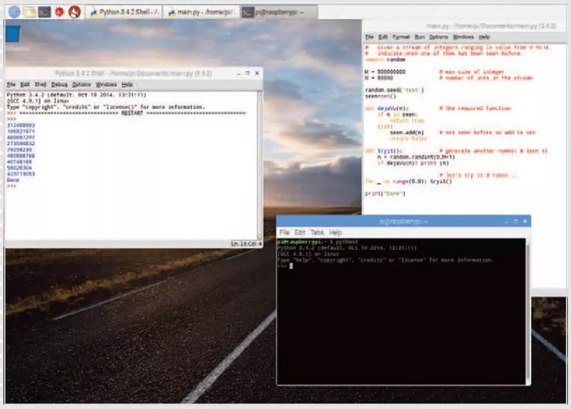
BOOKS

We have several great coding titles available via **www.pclpublications.com**. Our Pi books cover how to buy your first Raspberry Pi, set it up and use it; there are some great step-by-step project examples and guides to get the most from the Raspberry Pi too.

GET STARTED WITH PYTHON & C++ PROGRAMMING
Coding & Programming For Beginners
Jargon-free Tips & Advice
Step-by-step Tutorials
Clear Full Colour Guides
Over 440 Tips & Hints Inside
GET STARTED WITH CODING & PROGRAMMING
C++ & Python For Beginners
TRICKS & TIPS
Raspberry Pi
Advanced guides and tutorials for your Raspberry Pi 4, 400 and Raspberry Pico
OVER 485 SECRETS & HACKS
Next level secrets and fixes get to the heart of Python and Linux on your Pi
We share our awesome tips and projects for your Raspberry Pi
Everything you need to take your Pi coding project skills to the next level
NEW
FOR RASPBERRY Pi 4, 400 & PICO
The Complete Raspberry Pi Coding Manual
The essential manual to improve your Raspberry Pi coding skills
www.pclpublications.com
Papercut
100% INDEPENDENT

RASPBIAN

The Raspberry Pi's main operating system is a Debian-based Linux distribution that comes with everything you need in a simple to use package. It's streamlined for the Pi and is an ideal platform for hardware and software projects, Python programming and even as a desktop computer.


Python 3.4.2 Shell - /...
main.py - /home/pi/...
pi@raspberrypi: ~
File Edit Shell Debug Options Windows Help
File Edit Format Run Options Windows Help
File Edit Tabs Help

Getting to Know Python

Python is the greatest computer programming language ever created. It enables you to fully harness the power of a computer, in a language that's clean and easy to understand.

WHAT IS PROGRAMMING?

It helps to understand what a programming language is before you try to learn one, and Python is no different. Let's take a look at how Python came about and how it relates to other languages.

PYTHON

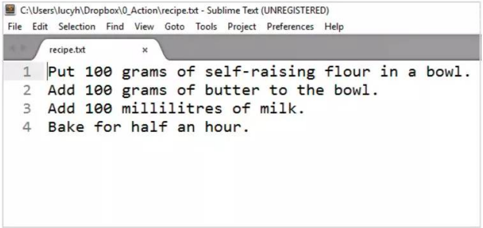
A programming language is a list of instructions that a computer follows. These instructions can be as simple as displaying your name or playing a music file, or as complex as building a whole virtual world. Python is a programming language conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language.

Guido van Rossum, the father of Python.

PROGRAMMING RECIPES

Programs are like recipes for computers. A recipe to bake a cake could go like this:

Put 100 grams of self-raising flour in a bowl.
Add 100 grams of butter to the bowl.
Add 100 millilitres of milk.
Bake for half an hour.

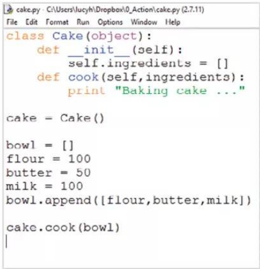

C:\Users\lucyh\Dropbox\0_Action\recipe.txt - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
recipe.txt

```
Put 100 grams of self-raising flour in a bowl.
Add 100 grams of butter to the bowl.
Add 100 millilitres of milk.
Bake for half an hour.
```

CODE

Just like a recipe, a program consists of instructions that you follow in order. A program that describes a cake might run like this:

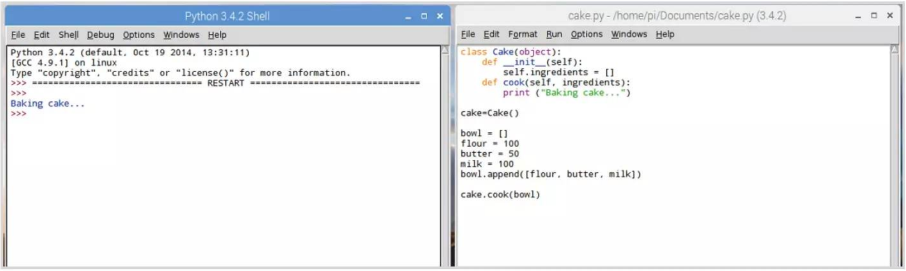
```
bowl = []
flour = 100
butter = 50
milk = 100
bowl.append([flour,butter,milk])
cake.cook(bowl)
```


cake.py - C:\Users\lucyh\Dropbox\0_Action\cake.py (2.7.11)
File Edit Format Run Options Window Help

```
class Cake(object):
    def __init__(self):
        self.ingredients = []
    def cook(self,ingredients):
        print "Baking cake ..."

cake = Cake()

bowl = []
flour = 100
butter = 50
milk = 100
bowl.append([flour,butter,milk])

cake.cook(bowl)
```

PROGRAM COMMANDS

You might not understand some of the Python commands, like bowl.append and cake.cook(bowl). The first is a list, the second an object; we'll look at both in this book. The main thing to know is that it's easy to read commands in Python. Once you learn what the commands do, it's easy to figure out how a program works.

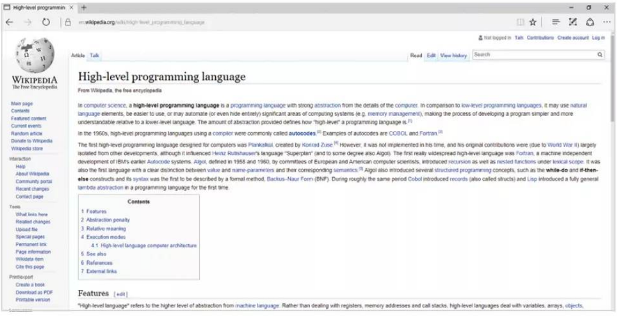

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
cake.py - /home/pi/Documents/cake.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Baking cake...
>>>
```

```
class Cake(object):
    def __init__(self):
        self.ingredients = []
    def cook(self, ingredients):
        print ("Baking cake...")

cake=Cake()

bowl = []
flour = 100
butter = 50
milk = 100
bowl.append([flour, butter, milk])

cake.cook(bowl)
```

HIGH-LEVEL LANGUAGES

Computer languages that are easy to read are known as "high-level". This is because they fly high above the hardware (also referred to as "the metal"). Languages that "fly close to the metal," like Assembly, are known as "low-level". Low-level languages commands read a bit like this: `msg db ,0xa len equ $ - msg`.



High-level programming language

From Wikipedia, the free encyclopedia

In computer science, a **high-level programming language** is a programming language with strong abstraction from the details of the computer. In comparison to low-level programming languages, it may use natural language elements, be easier to use, or may automate (or even hide entirely) significant areas of computing systems (e.g. memory management), making the process of developing a program simpler and more understandable relative to a lower-level language. The amount of abstraction provided defines how "high-level" a programming language is.[1]

In the 1960s, high-level programming languages using a compiler were commonly called **autocodes**.[2] Examples of autocodes are COBOL and Fortran.[3]

The first high-level programming language designed for computers was Plankalkül, created by Konrad Zuse.[4] However, it was not implemented in his time, and his original contributions were (due to World War II) largely isolated from other developments, although it influenced Heinz Rutishauser's language "Superplan" (and to some degree also Algol). The first really widespread high-level language was Fortran, a machine independent development of IBM's earlier Autocode systems. Algol, defined in 1958 and 1960, by committees of European and American computer scientists, introduced recursion as well as nested functions under lexical scope. It was also the first language with a clear distinction between value and name-parameters and their corresponding semantics.[5] Algol also introduced several structured programming concepts, such as the **while-do** and **if-then-else** constructs and its syntax was the first to be described by a formal method, Backus–Naur Form (BNF). During roughly the same period Cobol introduced records (also called structs) and Lisp introduced a fully general lambda abstraction in a programming language for the first time.

Contents

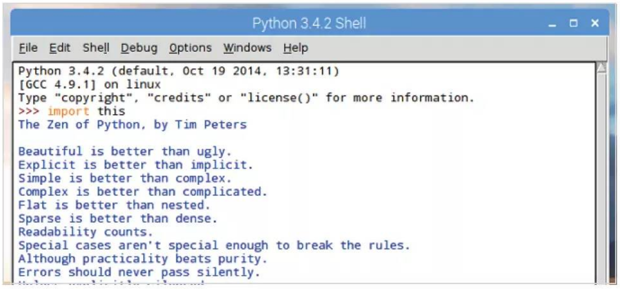
1 Features
2 Abstraction penalty
3 Relative meaning
4 Execution modes
  4.1 High-level language computer architecture
5 See also
6 References
7 External links

Features [edit]

"High-level language" refers to the higher level of abstraction from machine language. Rather than dealing with registers, memory addresses and call stacks, high-level languages deal with variables, arrays, objects,

ZEN OF PYTHON

Python lets you access all the power of a computer in a language that humans can understand. Behind all this is an ethos called "The Zen of Python." This is a collection of 20 software principles that influences the design of the language. Principles include "Beautiful is better than ugly" and "Simple is better than complex." Type `import this` into Python and it will display all the principles.



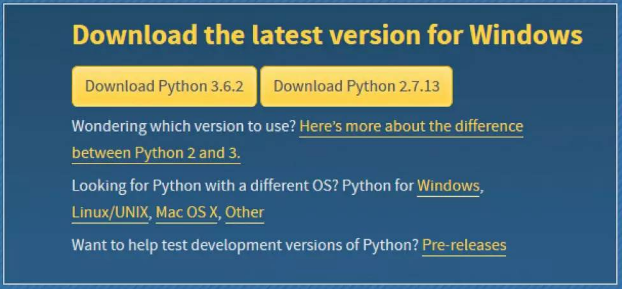
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
```

PYTHON 3 VS PYTHON 2

In a typical computing scenario, Python is complicated somewhat by the existence of two active versions of the language: Python 2 and Python 3.

WORLD OF PYTHON

When you visit the Python Download page you'll notice that there are two buttons available: one for Python 3.6.2 and the other for Python 2.7.13; correct at the time of writing (remember Python is frequently updated so you may see different version numbers).



Download the latest version for Windows

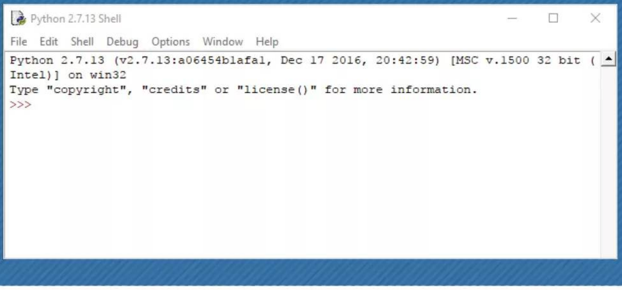
Wondering which version to use? Here's more about the difference between Python 2 and 3.

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

PYTHON 2.X

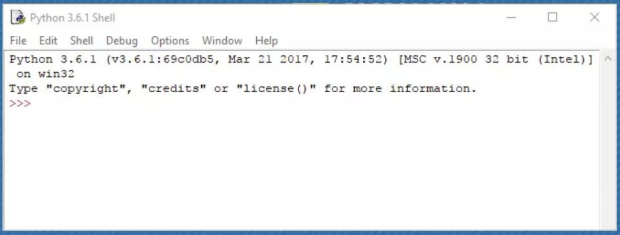
So why two? Well, Python 2 was originally launched in 2000 and has since then adopted quite a large collection of modules, scripts, users, tutorials and so on. Over the years Python 2 has fast become one of the first go to programming languages for beginners and experts to code in, which makes it an extremely valuable resource.



```
Python 2.7.13 (v2.7.13:a06454b1a1fa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (
Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

PYTHON 3.X

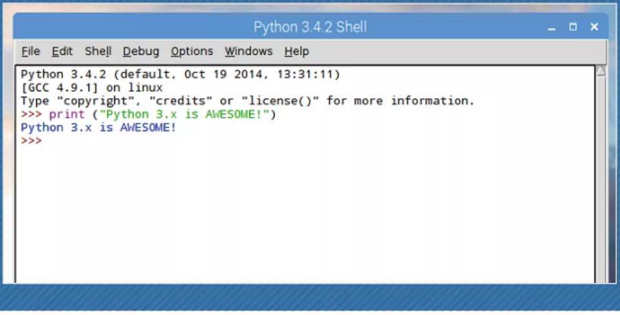
In 2008 Python 3 arrived with several new and enhanced features. These features provide a more stable, effective and efficient programming environment but sadly, most (if not all) of these new features are not compatible with Python 2 scripts, modules and tutorials. Whilst not popular at first, Python 3 has since become the cutting edge of Python programming.



```
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

3.X WINS

Python 3's growing popularity has meant that it's now prudent to start learning to develop with the new features and begin to phase out the previous version. Many development companies, such as SpaceX and NASA use Python 3 for snippets of important code.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print ("Python 3.x is AWESOME!")
Python 3.x is AWESOME!
>>>
```

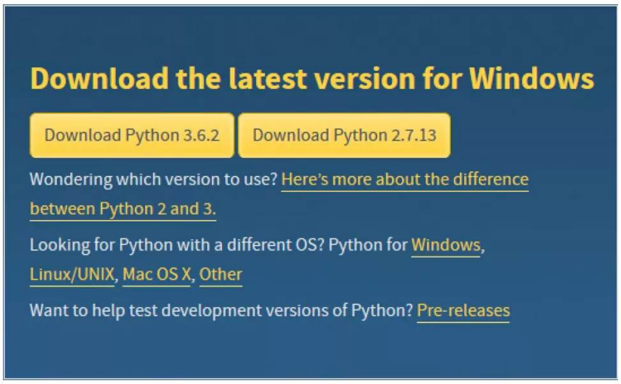
How to Set Up Python in Windows

Windows users can easily install the latest version of Python via the main Python Downloads page. Whilst most seasoned Python developers may shun Windows as the platform of choice for building their code, it's still an ideal starting point for beginners.

INSTALLING PYTHON 3.X

Microsoft Windows doesn't come with Python preinstalled as standard, so you're going to have to install it yourself manually. Thankfully, it's an easy process to follow.

STEP 1 Start by opening your web browser to **www.python.org/downloads/**. Look for the button detailing the download link for Python 3.x.x (in our case this is Python 3.6.2 but as mentioned you may see later versions of 3).



Download the latest version for Windows

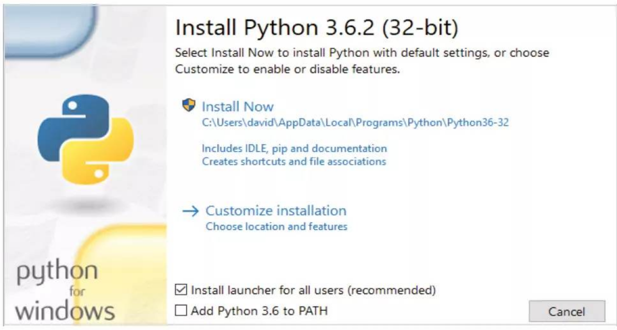
Download Python 3.6.2 Download Python 2.7.13

Wondering which version to use? Here's more about the difference between Python 2 and 3.

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

STEP 2 Click the download button for version 3.x, and save the file to your Downloads folder. When the file is downloaded, double-click the executable and the Python installation wizard will launch. From here you have two choices: Install Now and Customise Installation. We recommend opting for the Customise Installation link.



Install Python 3.6.2 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

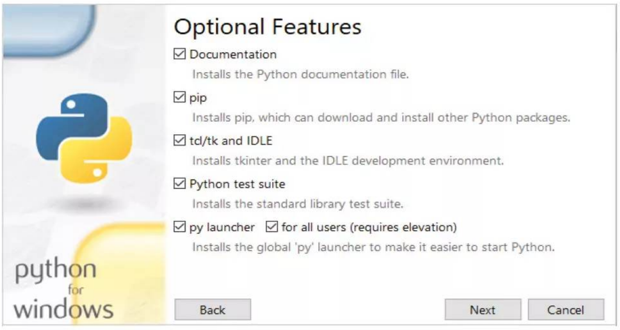
Install Now
C:\Users\david\AppData\Local\Programs\Python\Python36-32

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☐ Add Python 3.6 to PATH

STEP 3 Choosing the Customise option allows you to specify certain parameters, and whilst you may stay with the defaults, it's a good habit to adopt as sometimes (not with Python, thankfully) installers can include unwanted additional features. On the first screen available, ensure all boxes are ticked and click the Next button.



Optional Features

☑ Documentation
Installs the Python documentation file.

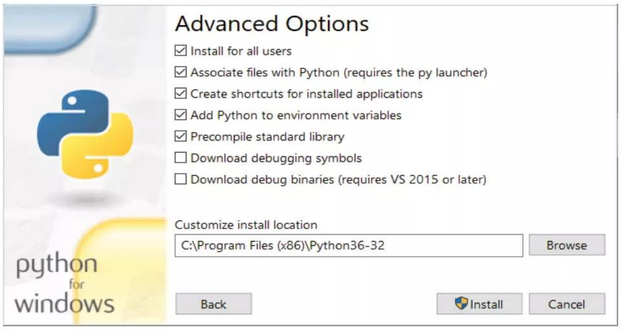
☑ pip
Installs pip, which can download and install other Python packages.

☑ tcl/tk and IDLE
Installs tkinter and the IDLE development environment.

☑ Python test suite
Installs the standard library test suite.

☑ py launcher ☑ for all users (requires elevation)
Installs the global 'py' launcher to make it easier to start Python.

STEP 4 The next page of options include some interesting additions to Python. Ensure the Associate file with Python, Create Shortcuts, Add Python to Environment Variables, Precompile Standard Library and Install For All Users options are ticked. These make using Python later much easier. Click Install when you're ready to continue.



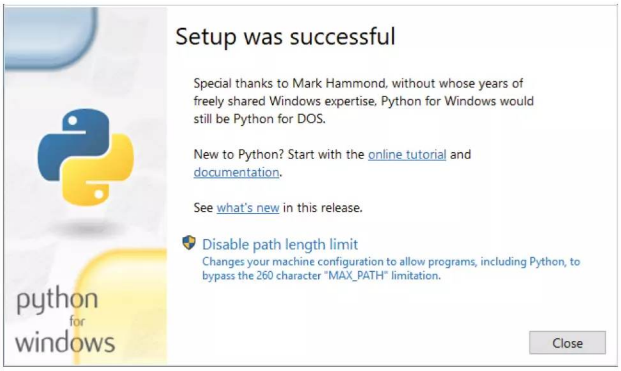
Advanced Options

☑ Install for all users
☑ Associate files with Python (requires the py launcher)
☑ Create shortcuts for installed applications
☑ Add Python to environment variables
☑ Precompile standard library
☐ Download debugging symbols
☐ Download debug binaries (requires VS 2015 or later)

Customize install location
C:\Program Files (x86)\Python36-32

STEP 5

You may need to confirm the installation with the Windows authentication notification. Simply click Yes and Python will begin to install. Once the installation is complete the final Python wizard page will allow you to view the latest release notes, and follow some online tutorials.



Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

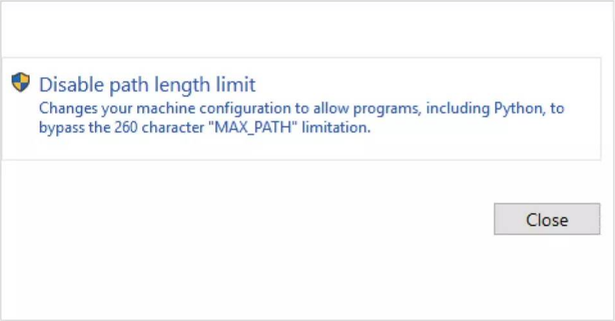
New to Python? Start with the online tutorial and documentation.

See what's new in this release.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

STEP 6

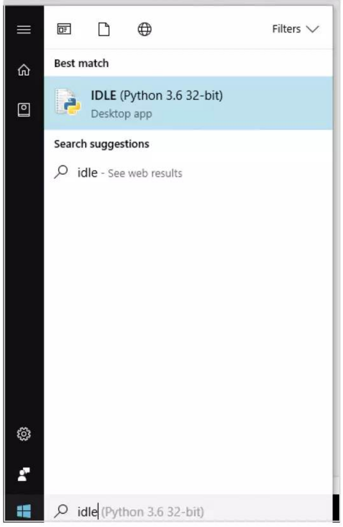
Before you close the install wizard window, however, it's best to click on the link next to the shield detailed Disable Path Length Limit. This will allow Python to bypass the Windows 260 character limitation, enabling you to execute Python programs stored in deep folders arrangements. Again, click Yes to authenticate the process; then you can Close the installation window.



Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

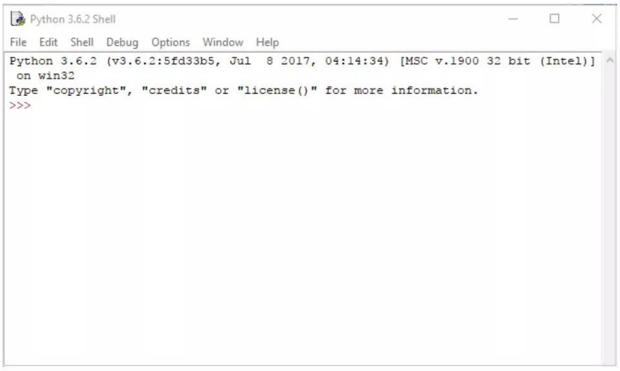
STEP 7

Windows 10 users will now find the installed Python 3.x within the Start button Recently Added section. The first link, Python 3.6 (32-bit) will launch the command line version of Python when clicked (more on that in a moment). To open the IDLE, type IDLE into Windows start.



STEP 8

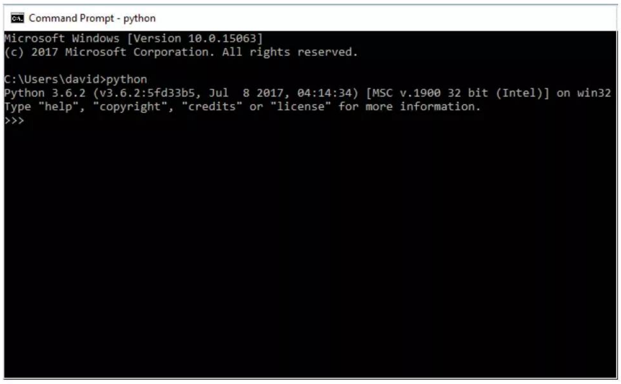
Clicking on the IDLE (Python 3.6 32-bit) link will launch the Python Shell, where you can begin your Python programming journey. Don't worry if your version is newer, as long as it's Python 3.x our code will work inside your Python 3 interface.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
```

STEP 9

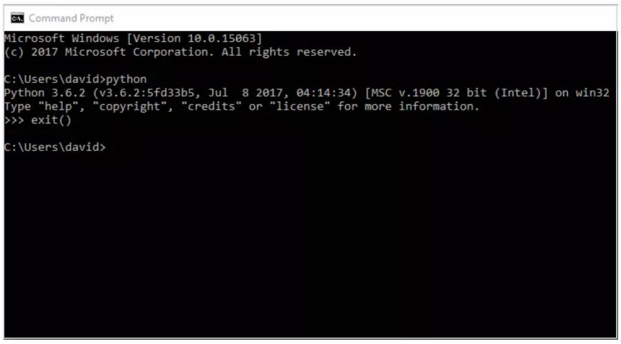
If you now click on the Windows Start button again, and this time type: `CMD`, you'll be presented with the Command Prompt link. Click it to get to the Windows command line environment. To enter Python within the command line, you need to type: `python` and press Enter.



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\david>python
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

STEP 10

The command line version of Python works in much the same way as the Shell you opened in Step 8; note the three left-facing arrows (>>>). Whilst it's a perfectly fine environment, it's not too user-friendly, so leave the command line for now. Enter: `exit()` to leave and close the Command Prompt window.



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\david>python
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()

C:\Users\david>
```

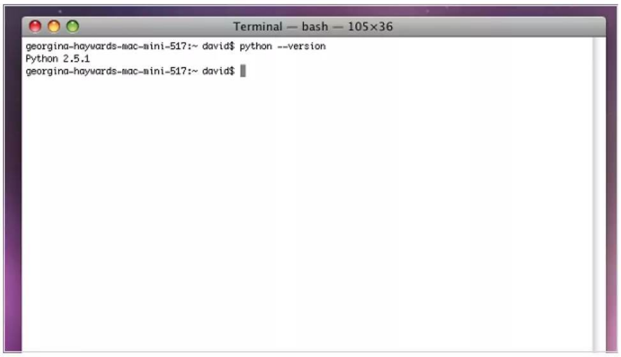
How to Set Up Python on a Mac

If you're running an Apple Mac, then setting up Python is incredibly easy. In fact a version of Python is already installed. However, you should make sure you're running the latest version.

INSTALLING PYTHON

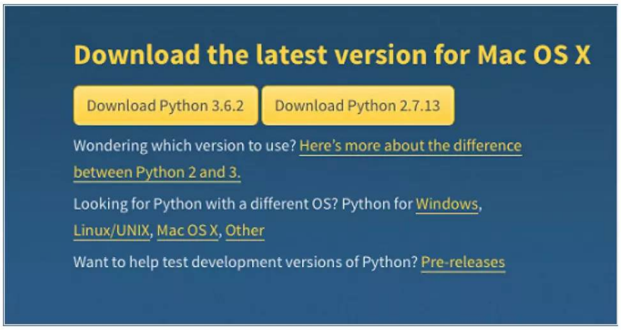
Apple's operating system comes with Python installed, so you don't need to install it separately. However, Apple doesn't update Python very often and you're probably running an older version. So it makes sense to check and update first.

STEP 1 Open a new Terminal window by clicking Go > Utilities, then double-click the Terminal icon. Now enter: `python --version`. You should see "Python 2.5.1" and even later, if Apple has updated the OS and Python installation. Either way, it's best to check for the latest version.



```
georgina-haywards-mac-mini-517:~ david$ python --version
Python 2.5.1
georgina-haywards-mac-mini-517:~ david$ 
```

STEP 2 Open Safari and head over to **www.python.org/downloads**. Just as with the Windows setup procedure on the previous pages, you can see two yellow download buttons: one for Python 3.6.2, and the other for Python 2.7.13. Note, that version numbers may be different due to the frequent releases of Python.



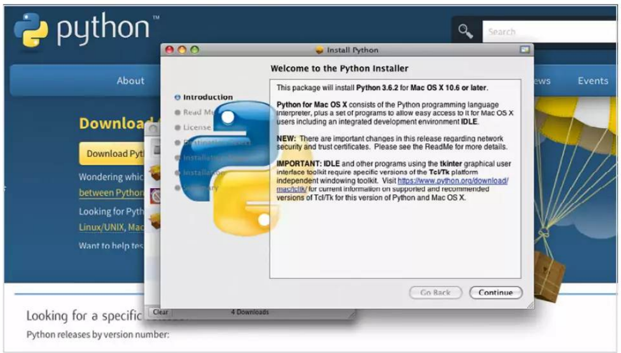
Download the latest version for Mac OS X

Wondering which version to use? Here's more about the difference between Python 2 and 3.

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases

STEP 3 Click on the latest version of Python 3.x, in our case this is the download button for Python 3.6.2. This will automatically download the latest version of Python and depending on how you've got your Mac configured, it automatically starts the installation wizard.



Welcome to the Python Installer

This package will install **Python 3.6.2** for **Mac OS X 10.6 or later**.

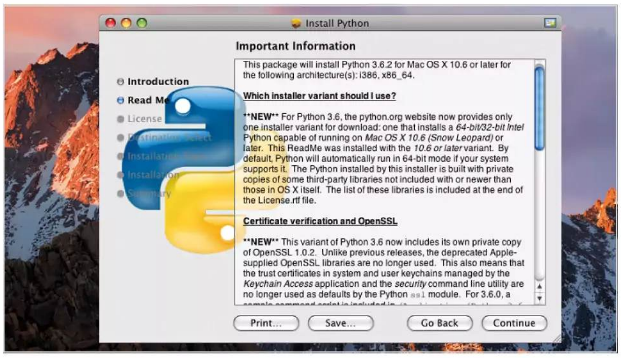
Python for Mac OS X consists of the Python programming language interpreter, plus a set of programs to allow easy access to it for Mac OS X users including an integrated development environment **IDLE**.

NEW: There are important changes in this release regarding network security and trust certificates. Please see the ReadMe for more details.

IMPORTANT: IDLE and other programs using the **tkinter** graphical user interface toolkit require specific versions of the **Tcl/Tk** platform independent windowing toolkit. Visit https://www.python.org/download/mac/tcltk/ for current information on supported and recommended versions of Tcl/Tk for this version of Python and Mac OS X.

Looking for a specific release?
Python releases by version number:

STEP 4 With the Python installation wizard open, click on the Continue button to begin the installation. It's worth taking a moment to read through the Important Information section, in case it references something that applies to your version of macOS. When ready, click Continue again.



Important Information

This package will install Python 3.6.2 for Mac OS X 10.6 or later for the following architecture(s): i386, x86_64.

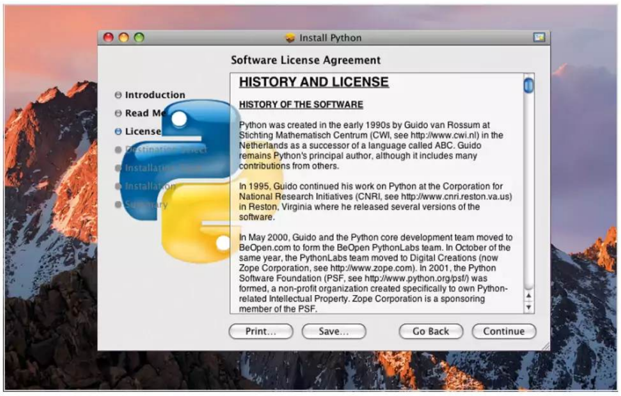
Which installer variant should I use?

NEW For Python 3.6, the python.org website now provides only one installer variant for download: one that installs a *64-bit/32-bit Intel* Python capable of running on *Mac OS X 10.6 (Snow Leopard)* or later. This ReadMe was installed with the *10.6 or later* variant. By default, Python will automatically run in 64-bit mode if your system supports it. The Python installed by this installer is built with private copies of some third-party libraries not included with or newer than those in OS X itself. The list of these libraries is included at the end of the License.rtf file.

Certificate verification and OpenSSL

NEW This variant of Python 3.6 now includes its own private copy of OpenSSL 1.0.2. Unlike previous releases, the deprecated Apple-supplied OpenSSL libraries are no longer used. This also means that the trust certificates in system and user keychains managed by the *Keychain Access* application and the *security* command line utility are no longer used as defaults by the Python ssl module. For 3.6.0, a

STEP 5 The next section details the Software License Agreement, and whilst not particularly interesting to most folks, it's probably worth a read. When you're ready, click on the Continue button once again.


Install Python
Introduction
Read Me
License
Destination Select
Installation Type
Installation
Summary
Print...
Save...
Go Back
Continue

Software License Agreement

HISTORY AND LICENSE

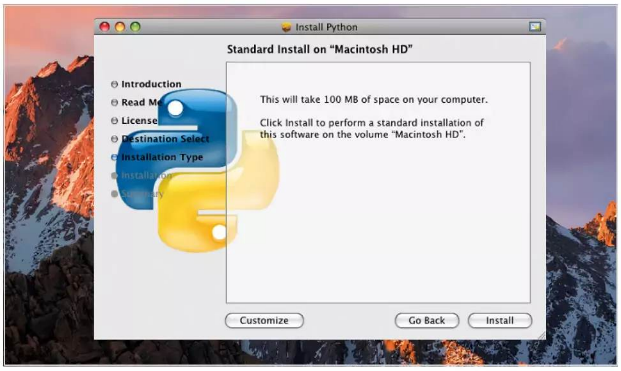
HISTORY OF THE SOFTWARE

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see http://www.cwi.nl) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see http://www.cnri.reston.va.us) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation, see http://www.zope.com). In 2001, the Python Software Foundation (PSF, see http://www.python.org/psf/) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

STEP 6 Finally you're be presented with the amount of space Python will take up on your system and an Install button, which you need to click to start the actual installation of Python 3.x on to your Mac. You may need to enter your password to authenticate the installation process.

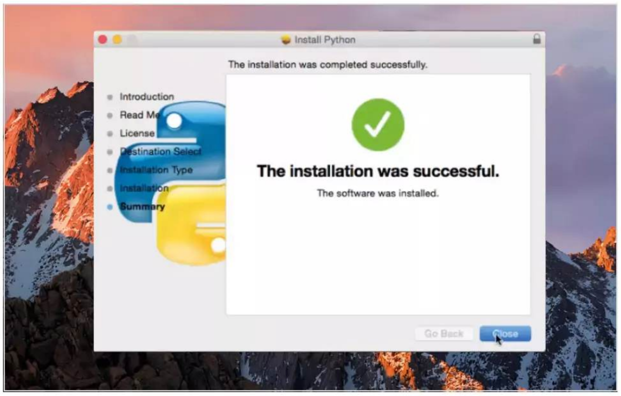

Install Python
Introduction
Read Me
License
Destination Select
Installation Type
Installation
Summary
Customize
Go Back
Install

Standard Install on "Macintosh HD"

This will take 100 MB of space on your computer.

Click Install to perform a standard installation of this software on the volume "Macintosh HD".

STEP 7 The installation shouldn't take too long; the older Mac Mini we used in this section is a little slower than more modern Mac machines and it only took around thirty seconds for the Installation Successful prompt to be displayed.

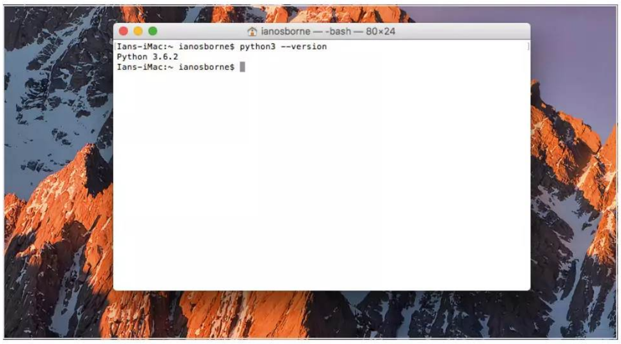

Install Python
Introduction
Read Me
License
Destination Select
Installation Type
Installation
Summary
Go Back
Close

The installation was completed successfully.

The installation was successful.

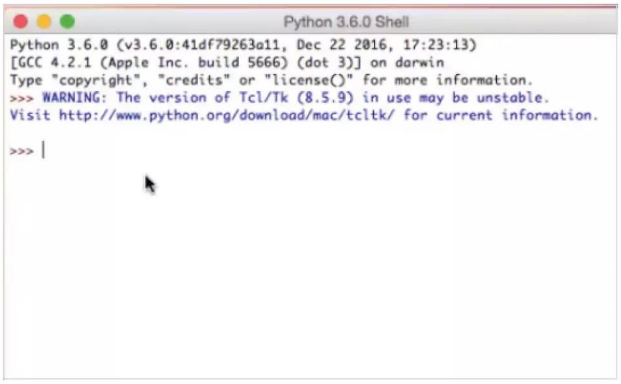
The software was installed.

STEP 8 There's nothing much else left to do in the Python installation wizard so you can click the Close button. If you now drop back into a Terminal session and re-enter the command: `python3 --version`, you can see the new version is now listed. To enter the command line version of Python, you need to enter: `python3`. To exit, it's: `exit()`.


ianosborne — -bash — 80×24

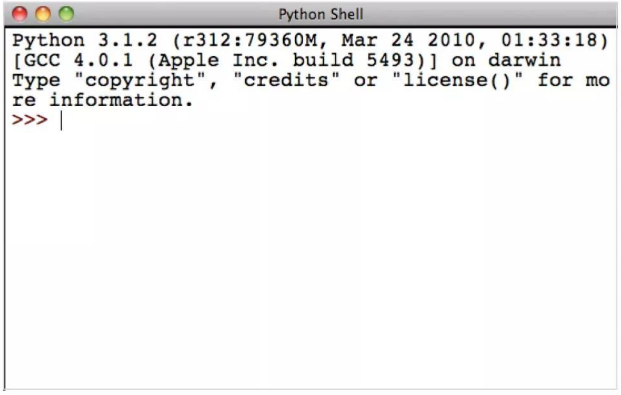
```
Ians-iMac:~ ianosborne$ python3 --version
Python 3.6.2
Ians-iMac:~ ianosborne$ 
```

STEP 9 You need to search in Finder for the Python IDLE; when you've found it, click it to launch and it should look similar to that of the Windows IDLE version shown on the previous page. The only difference being the Mac detected hardware platform it's running on.


Python 3.6.0 Shell

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 22 2016, 17:23:13)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> WARNING: The version of Tcl/Tk (8.5.9) in use may be unstable.
Visit http://www.python.org/download/mac/tcltk/ for current information.

>>> |
```

STEP 10 Older Mac versions may have trouble with the newer versions of Python, in which case you will need to revert to a previous Python 3.x build; as long as you're using Python 3.x, the code in this book will work for you.


Python Shell

```
Python 3.1.2 (r312:79360M, Mar 24 2010, 01:33:18)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "copyright", "credits" or "license()" for mo
re information.
>>> |
```

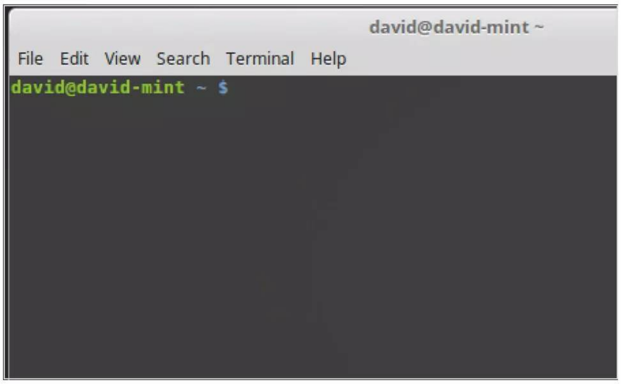
How to Set Up Python in Linux

Python version 2.x is already installed in most Linux distributions but as we're going to be using Python 3.x, there's a little work we need to do first to get hold of it. Thankfully, it's not too difficult.

PYTHON PENGUIN

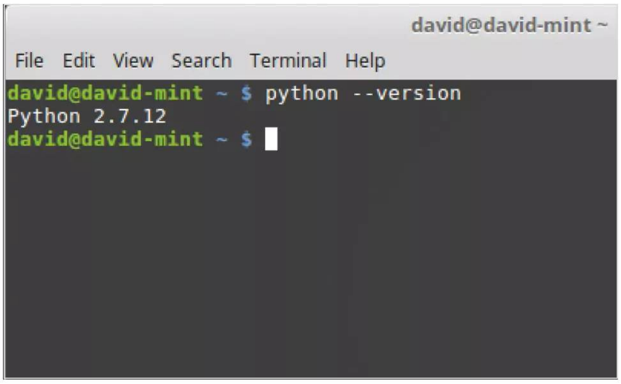
Linux is such a versatile operating system that it's often difficult to nail down just one way of doing something. Different distributions go about installing software in different ways, so we will stick to Linux Mint 18.1 for this particular tutorial.

STEP 1 First you need to ascertain which version of Python is currently installed in your Linux system; as we mentioned, we're going to be using Linux Mint 18.1 for this section. As with macOS, drop into a Terminal by pressing Ctrl+Atl+T.



```
david@david-mint ~ $
```

STEP 2 Next enter: `python --version` into the Terminal screen. You should have the output relating to version 2.x of Python in the display. Ours in this particular case is Python 2.7.12.

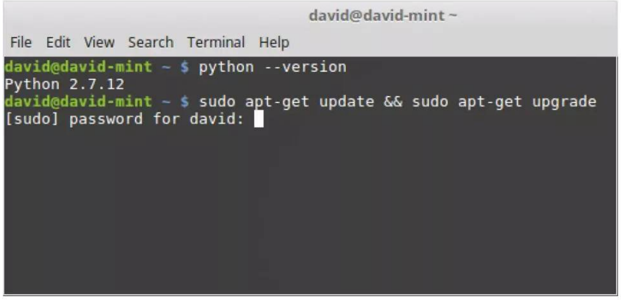


```
david@david-mint ~ $ python --version
Python 2.7.12
david@david-mint ~ $
```

STEP 3 Some Linux distros will automatically update the installation of Python to the latest versions whenever the system is updated. To check, first do a system update and upgrade with:

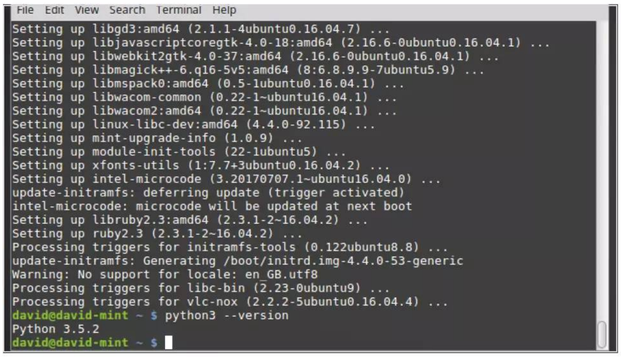
```
sudo apt-get update && sudo apt-get upgrade
```

Enter your password and let the system do any updates.



```
david@david-mint ~ $ python --version
Python 2.7.12
david@david-mint ~ $ sudo apt-get update && sudo apt-get upgrade
[sudo] password for david:
```

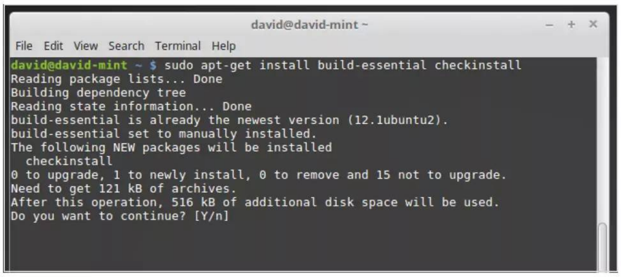
STEP 4 Once the update and upgrade is complete, you may need to answer 'Y' to authorise any upgrades, enter: `python3 --version` to see if Python 3.x is updated or even installed. In the case of Linux Mint, the version we have is Python 3.5.2, which is fine for our purposes.



```
Setting up libgd3:amd64 (2.1.1-4ubuntu0.16.04.7) ...
Setting up libjavascriptcoregtk-4.0-18:amd64 (2.16.6-0ubuntu0.16.04.1) ...
Setting up libwebkit2gtk-4.0-37:amd64 (2.16.6-0ubuntu0.16.04.1) ...
Setting up libmagick++-6.q16-5v5:amd64 (8:6.8.9.9-7ubuntu5.9) ...
Setting up libmspack0:amd64 (0.5-1ubuntu0.16.04.1) ...
Setting up libwacom-common (0.22-1~ubuntu16.04.1) ...
Setting up libwacom2:amd64 (0.22-1~ubuntu16.04.1) ...
Setting up linux-libc-dev:amd64 (4.4.0-92.115) ...
Setting up mint-upgrade-info (1.0.9) ...
Setting up module-init-tools (22-1ubuntu5) ...
Setting up xfonts-utils (1:7.7+3ubuntu0.16.04.2) ...
Setting up intel-microcode (3.20170707.1~ubuntu16.04.0) ...
update-initramfs: deferring update (trigger activated)
intel-microcode: microcode will be updated at next boot
Setting up libruby2.3:amd64 (2.3.1-2~16.04.2) ...
Setting up ruby2.3 (2.3.1-2~16.04.2) ...
Processing triggers for initramfs-tools (0.122ubuntu8.8) ...
update-initramfs: Generating /boot/initrd.img-4.4.0-53-generic
Warning: No support for locale: en_GB.utf8
Processing triggers for libc-bin (2.23-0ubuntu9) ...
Processing triggers for vlc-nox (2.2.2-5ubuntu0.16.04.4) ...
david@david-mint ~ $ python3 --version
Python 3.5.2
david@david-mint ~ $
```

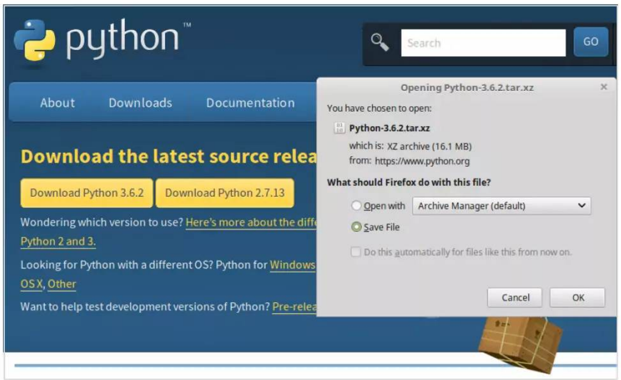
STEP 5 However, if you want the latest version, 3.6.2 as per the Python website at the time of writing, you need to build Python from source. Start by entering these commands into the Terminal:

```
sudo apt-get install build-essential checkinstall
sudo apt-get install libreadline-gplv2-dev
libncursesw5-dev libssl-dev libsqlite3-dev tk-dev
libgdbm-dev libc6-dev libbz2-dev
```



```
david@david-mint ~ $ sudo apt-get install build-essential checkinstall
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.1ubuntu2).
build-essential set to manually installed.
The following NEW packages will be installed
  checkinstall
0 to upgrade, 1 to newly install, 0 to remove and 15 not to upgrade.
Need to get 121 kB of archives.
After this operation, 516 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

STEP 6 Open up your Linux web browser and go to the Python download page: **www.python.org/downloads**. Click on the Download Python 3.6.2 (or whichever version it's on when you look) to download the source Python-3.6.2.tar.xz file.



Download the latest source relea

Wondering which version to use? Here's more about the diff Python 2 and 3.

Looking for Python with a different OS? Python for Windows OS X, Other

Want to help test development versions of Python? Pre-relea

Opening Python-3.6.2.tar.xz

You have chosen to open:

Python-3.6.2.tar.xz

which is: XZ archive (16.1 MB)

from: https://www.python.org

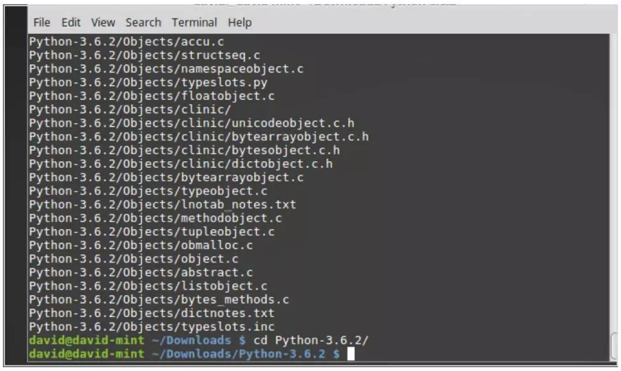
What should Firefox do with this file?

○ Open with Archive Manager (default)

● Save File

☐ Do this automatically for files like this from now on.

STEP 7 In the Terminal, go the Downloads folder by entering: `cd Downloads/`. Then unzip the contents of the downloaded Python source code with: `tar -xvf Python-3.6.2.tar.xz`. Now enter the newly unzipped folder with `cd Python-3.6.2/`.

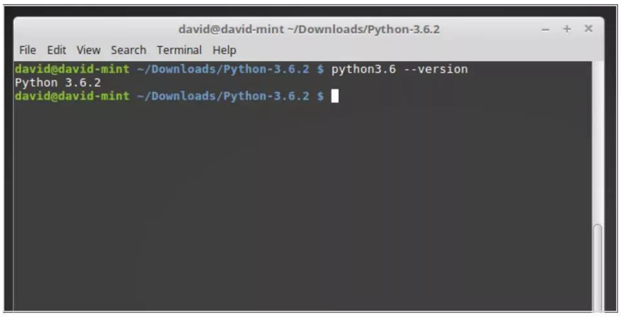


```
Python-3.6.2/Objects/accu.c
Python-3.6.2/Objects/structseq.c
Python-3.6.2/Objects/namespaceobject.c
Python-3.6.2/Objects/typeslots.py
Python-3.6.2/Objects/floatobject.c
Python-3.6.2/Objects/clinic/
Python-3.6.2/Objects/clinic/unicodeobject.c.h
Python-3.6.2/Objects/clinic/bytearrayobject.c.h
Python-3.6.2/Objects/clinic/bytesobject.c.h
Python-3.6.2/Objects/clinic/dictobject.c.h
Python-3.6.2/Objects/bytearrayobject.c
Python-3.6.2/Objects/typeobject.c
Python-3.6.2/Objects/lnotab_notes.txt
Python-3.6.2/Objects/methodobject.c
Python-3.6.2/Objects/tupleobject.c
Python-3.6.2/Objects/obmalloc.c
Python-3.6.2/Objects/object.c
Python-3.6.2/Objects/abstract.c
Python-3.6.2/Objects/listobject.c
Python-3.6.2/Objects/bytes_methods.c
Python-3.6.2/Objects/dictnotes.txt
Python-3.6.2/Objects/typeslots.inc
david@david-mint ~/Downloads $ cd Python-3.6.2/
david@david-mint ~/Downloads/Python-3.6.2 $
```

STEP 8 Within the Python folder, enter:

```
./configure
sudo make altinstall
```

This could a little while depending on the speed of your computer. Once finished, enter: `python3.6 --version` to check the installed latest version.

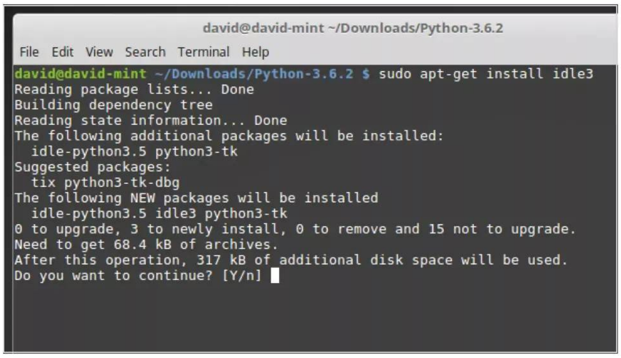

david@david-mint ~/Downloads/Python-3.6.2
File Edit View Search Terminal Help

```
david@david-mint ~/Downloads/Python-3.6.2 $ python3.6 --version
Python 3.6.2
david@david-mint ~/Downloads/Python-3.6.2 $
```

STEP 9 For the GUI IDLE, you need to enter the following command into the Terminal:

```
sudo apt-get install idle3
```

The IDLE can then be started with the command: `idle3`. Note, that IDLE runs a different version from the one you installed from source.


david@david-mint ~/Downloads/Python-3.6.2
File Edit View Search Terminal Help

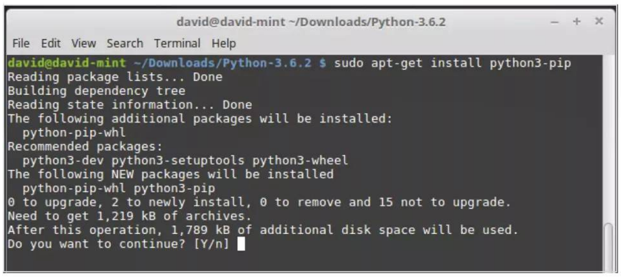
```
david@david-mint ~/Downloads/Python-3.6.2 $ sudo apt-get install idle3
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  idle-python3.5 python3-tk
Suggested packages:
  tix python3-tk-dbg
The following NEW packages will be installed
  idle-python3.5 idle3 python3-tk
0 to upgrade, 3 to newly install, 0 to remove and 15 not to upgrade.
Need to get 68.4 kB of archives.
After this operation, 317 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

STEP 10 You also need PIP (Pip Installs Packages) which is a tool to help you install more modules and extras. Enter: `sudo apt-get install python3-pip`

PIP is then installed; check for the latest update with:

```
pip3 install --upgrade pip
```

When complete, close the Terminal and Python 3.x will be available via the Programming section in your distro's menu.


david@david-mint ~/Downloads/Python-3.6.2
File Edit View Search Terminal Help

```
david@david-mint ~/Downloads/Python-3.6.2 $ sudo apt-get install python3-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  python-pip-whl
Recommended packages:
  python3-dev python3-setuptools python3-wheel
The following NEW packages will be installed
  python-pip-whl python3-pip
0 to upgrade, 2 to newly install, 0 to remove and 15 not to upgrade.
Need to get 1,219 kB of archives.
After this operation, 1,789 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Getting Started with Python

Getting started with Python may seem a little daunting at first but the language has been designed with simplicity in mind. Like most things, you need to start slowly, learn how to get a result and how to get what you want from the code.

In this section, we cover variables, numbers and expressions, user input, conditions and loops; and the types of errors you may well come across in your time with Python.

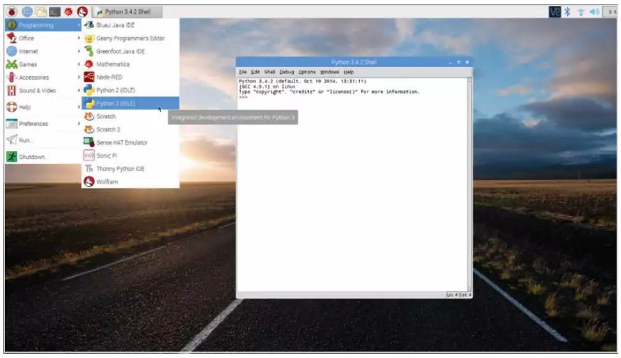
Starting Python for the First Time

We're going to be using the Raspberry Pi as our Python 3 hardware platform. The latest version of Raspbian comes preinstalled with Python 3, version 3.4.2 to be exact, so as long as you have a version 3 Shell, all our code will work.

STARTING PYTHON

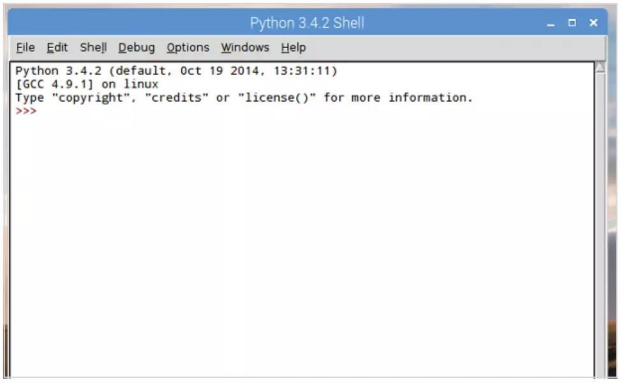
We're not going to go into the details of getting the Raspberry Pi up and running, there's plenty of material already available on that subject. However, once you're ready, fire up your Pi and get ready for coding.

STEP 1 With the Raspbian desktop loaded, click on the Menu button followed by Programming > Python 3 (IDLE). This opens the Python 3 Shell. Windows and Mac users can find the Python 3 IDLE Shell from within the Windows Start button menu and via Finder.


Python 3.4.2 Shell
Programming
Office
Internet
Games
Accessories
Sound & Video
Help
Preferences
Run...
Shutdown...
BlueJ Java IDE
Geany Programmer's Editor
Greenfoot Java IDE
Mathematica
Node-RED
Python 2 (IDLE)
Python 3 (IDLE)
Scratch
Scratch 2
Sense HAT Emulator
Sonic Pi
Thonny Python IDE
Wolfram
Integrated development environment for Python 3
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

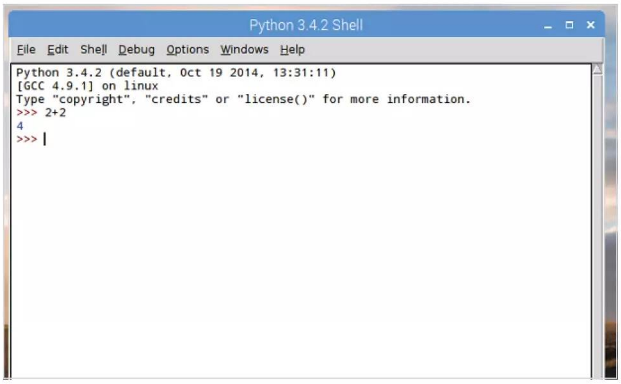
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

STEP 2 The Shell is where you can enter code and see the responses and output of code you've programmed into Python. This is a kind of sandbox, where you're able to try out some simple code and processes.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

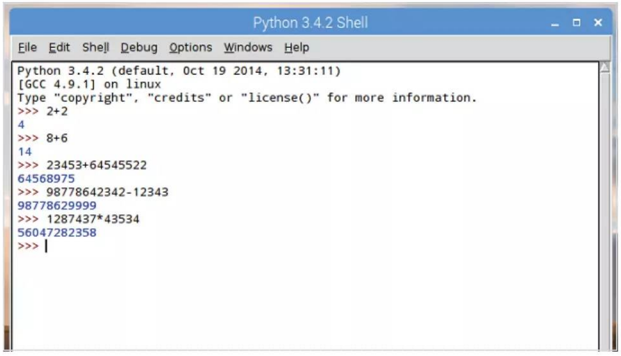
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

STEP 3 For example, in the Shell enter: 2+2 After pressing Enter, the next line displays the answer: 4. Basically, Python has taken the 'code' and produced the relevant output.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> |
```

STEP 4 The Python Shell acts very much like a calculator, since code is basically a series of mathematical interactions with the system. Integers, which are the infinite sequence of whole numbers can easily be added, subtracted, multiplied and so on.

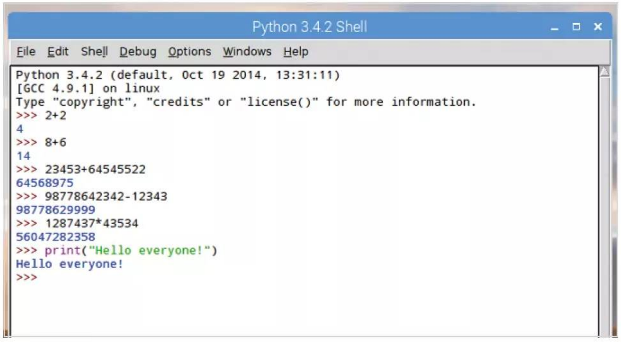

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 8+6
14
>>> 23453+64545522
64568975
>>> 98778642342-12343
98778629999
>>> 1287437*43534
56047282358
>>> |
```

STEP 5 While that's very interesting, it's not particularly exciting. Instead, try this:

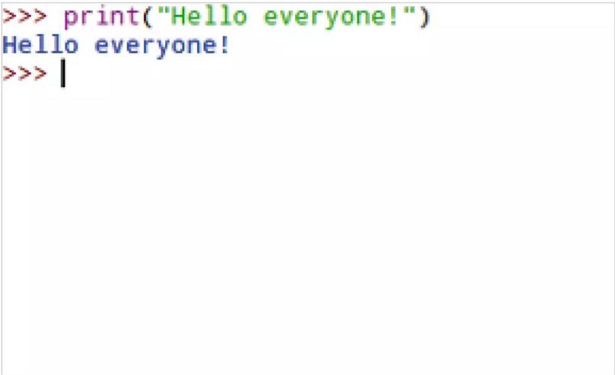
```
print("Hello everyone!")
```

Just like the code we entered in Sublime in the Installing a Text Editor section of this book.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 8+6
14
>>> 23453+64545522
64568975
>>> 98778642342-12343
98778629999
>>> 1287437*43534
56047282358
>>> print("Hello everyone!")
Hello everyone!
>>>
```

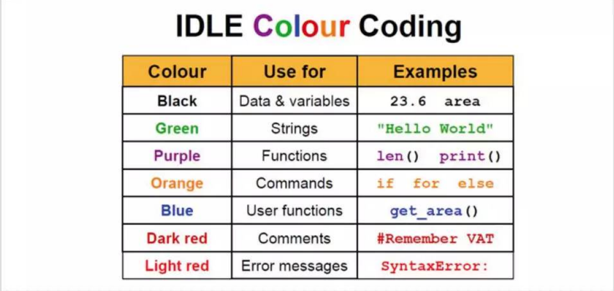
STEP 6 This is a little more like it, since you've just produced your first bit of code. The Print command is fairly self-explanatory, it prints things. Python 3 requires the brackets as well as quote marks in order to output content to the screen, in this case the 'Hello everyone!' bit.



```
>>> print("Hello everyone!")
Hello everyone!
>>> |
```

STEP 7 You may have noticed the colour coding within the Python IDLE. The colours represent different elements of Python code. They are:

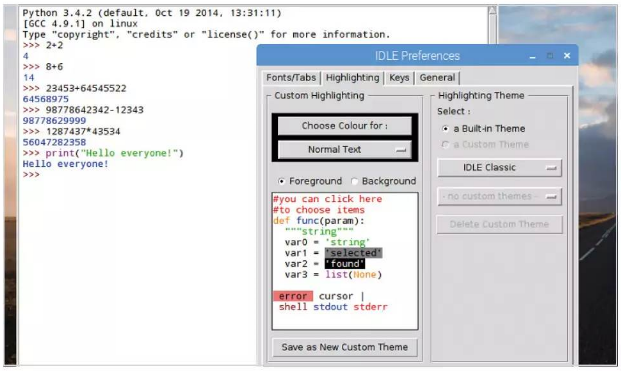
Black – Data and Variables
Green – Strings
Purple – Functions
Orange – Commands
Blue – User Functions
Dark Red – Comments
Light Red – Error Messages



IDLE Colour Coding

Colour	Use for	Examples
Black	Data & variables	`23.6 area`
Green	Strings	`"Hello World"`
Purple	Functions	`len() print()`
Orange	Commands	`if for else`
Blue	User functions	`get_area()`
Dark red	Comments	`#Remember VAT`
Light red	Error messages	`SyntaxError:`

STEP 8 The Python IDLE is a configurable environment. If you don't like the way the colours are represented, then you can always change them via Options > Configure IDLE and clicking on the Highlighting tab. However, we don't recommend that, as you won't be seeing the same as our screenshots.

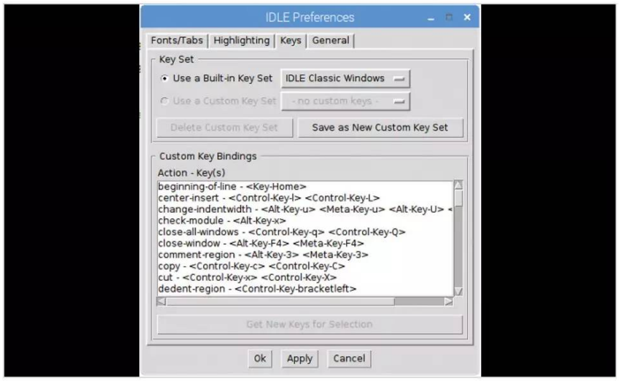


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 8+6
14
>>> 23453+64545522
64568975
>>> 98778642342-12343
98778629999
>>> 1287437*43534
56047282358
>>> print("Hello everyone!")
Hello everyone!
>>>
```

Select :
● a Built-in Theme
○ a Custom Theme
IDLE Classic
● Foreground ○ Background

```
#you can click here
#to choose items
def func(param):
  """string"""
  var0 = 'string'
  var1 = 'selected'
  var2 = 'found'
  var3 = list(None)

 error  cursor |
 shell stdout stderr
```

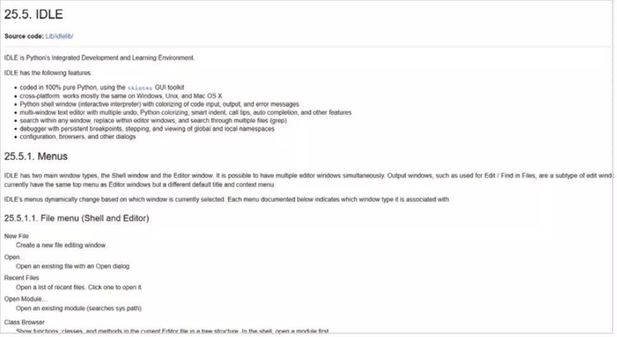
STEP 9 Just like most programs available, regardless of the operating system, there are numerous shortcut keys available. We don't have room for them all here but within the Options > Configure IDLE and under the Keys tab, you can see a list of the current bindings.



● Use a Built-in Key Set: IDLE Classic Windows
○ Use a Custom Key Set: - no custom keys -

Action - Key(s)
beginning-of-line - <Key-Home>
center-insert - <Control-Key-l> <Control-Key-L>
change-indentwidth - <Alt-Key-u> <Meta-Key-u> <Alt-Key-U> <
check-module - <Alt-Key-x>
close-all-windows - <Control-Key-q> <Control-Key-Q>
close-window - <Alt-Key-F4> <Meta-Key-F4>
comment-region - <Alt-Key-3> <Meta-Key-3>
copy - <Control-Key-c> <Control-Key-C>
cut - <Control-Key-x> <Control-Key-X>
dedent-region - <Control-Key-bracketleft>

STEP 10 The Python IDLE is a power interface and one that's actually been written in Python using one of the available GUI toolkits. If you want to know the many ins and outs of the Shell, we recommend you take a few moments to view **www.docs.python.org/3/library/idle.html**, which details many of the IDLE's features.



25.5. IDLE

Source code: Lib/idlelib/

IDLE is Python's Integrated Development and Learning Environment.

IDLE has the following features:

- coded in 100% pure Python, using the `tkinter` GUI toolkit
- cross-platform: works mostly the same on Windows, Unix, and Mac OS X
- Python shell window (interactive interpreter) with colorizing of code input, output, and error messages
- multi-window text editor with multiple undo, Python colorizing, smart indent, call tips, auto completion, and other features
- search within any window, replace within editor windows, and search through multiple files (grep)
- debugger with persistent breakpoints, stepping, and viewing of global and local namespaces
- configuration, browsers, and other dialogs

25.5.1. Menus

IDLE has two main window types, the Shell window and the Editor window. It is possible to have multiple editor windows simultaneously. Output windows, such as used for Edit / Find in Files, are a subtype of edit wind currently have the same top menu as Editor windows but a different default title and context menu.

IDLE's menus dynamically change based on which window is currently selected. Each menu documented below indicates which window type it is associated with.

25.5.1.1. File menu (Shell and Editor)

New File
  Create a new file editing window
Open...
  Open an existing file with an Open dialog
Recent Files
  Open a list of recent files. Click one to open it
Open Module...
  Open an existing module (searches sys.path)
Class Browser
  Show functions, classes, and methods in the current Editor file in a tree structure. In the shell, open a module first.

Your First Code

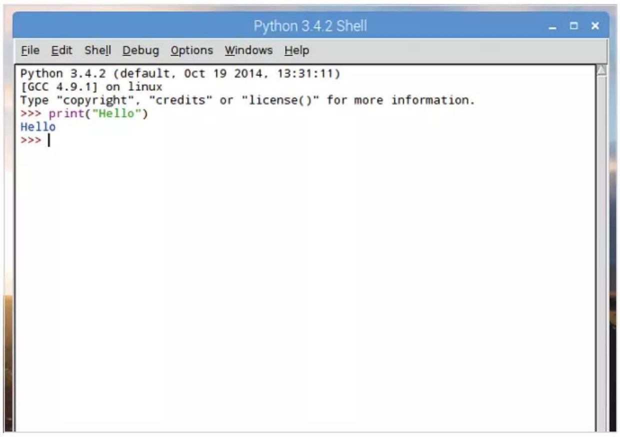
Essentially, you've already written your first piece of code with the 'print("Hello everyone!")' function from the previous tutorial. However, let's expand that and look at entering your code and playing around with some other Python examples.

PLAYING WITH PYTHON

With most languages, computer or human, it's all about remembering and applying the right words to the right situation. You're not born knowing these words, so you need to learn them.

STEP 1 If you've closed Python 3 IDLE, reopen it in whichever operating system version you prefer. In the Shell, enter the familiar following:

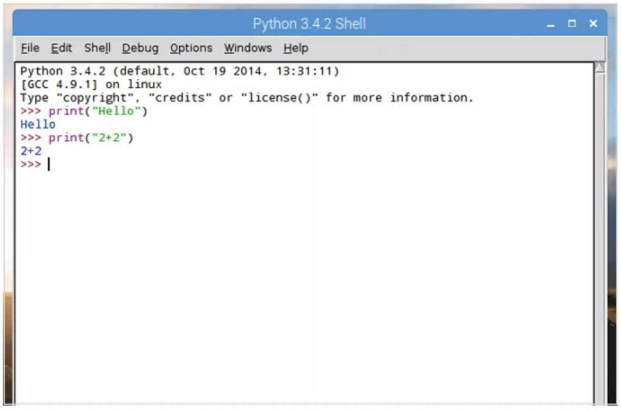
```
print(“Hello”)
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>>
```

STEP 2 Just as predicted, the word Hello appears in the Shell as blue text, indicating output from a string. It's fairly straightforward and doesn't require too much explanation. Now try:

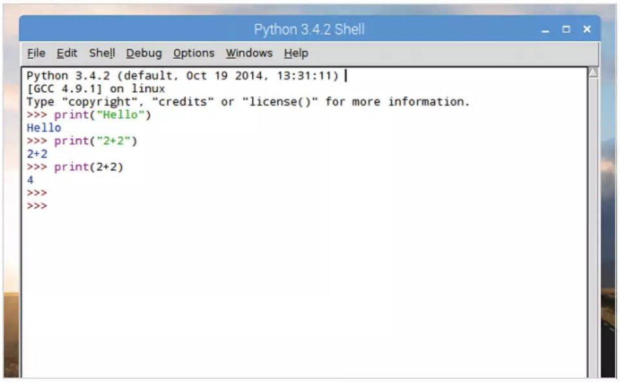
```
print(“2+2”)
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>>
```

STEP 3 You can see that instead of the number 4, the output is the 2+2 you asked to be printed to the screen. The quotation marks are defining what's being outputted to the IDLE Shell; to print the total of 2+2 you need to remove the quotes:

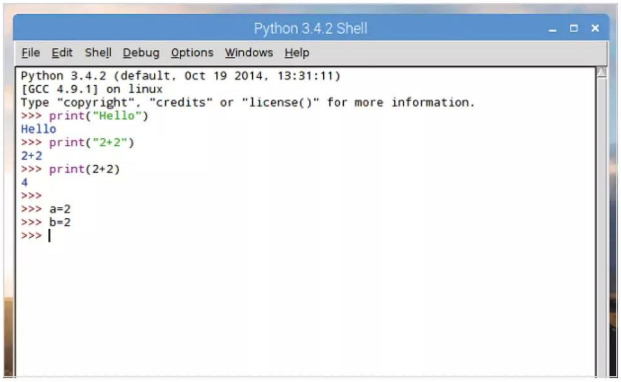
```
print(2+2)
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>>
>>>
```

STEP 4 You can continue as such, printing 2+2, 464+2343 and so on to the Shell. An easier way is to use a variable, which is something we will cover in more depth later. For now, enter:

```
a=2
b=2
```

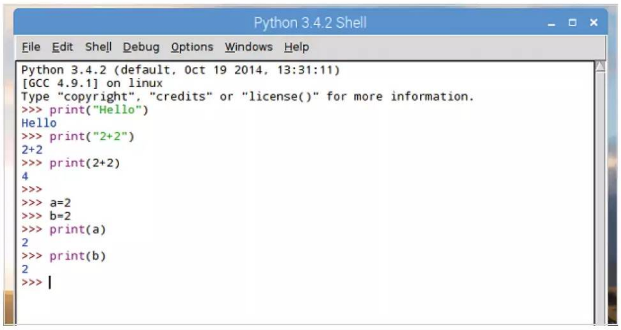

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>>
>>> a=2
>>> b=2
>>>
```

STEP 5

What you have done here is assign the letters a and b two values: 2 and 2. These are now variables, which can be called upon by Python to output, add, subtract, divide and so on for as long as their numbers stay the same. Try this:

```
print(a)
print(b)
```



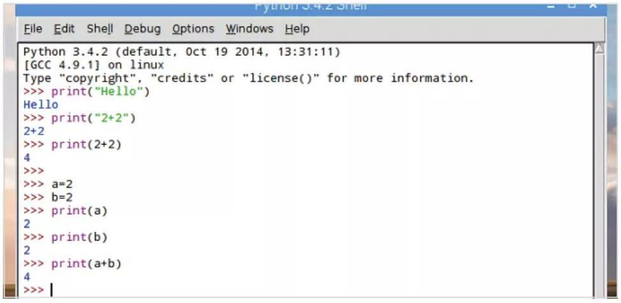
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>>
>>> a=2
>>> b=2
>>> print(a)
2
>>> print(b)
2
>>> |
```

STEP 6

The output of the last step displays the current values of both a and b individually, as you've asked them to be printed separately. If you want to add them up, you can use the following:

```
print(a+b)
```

This code simply takes the values of a and b, adds them together and outputs the result.

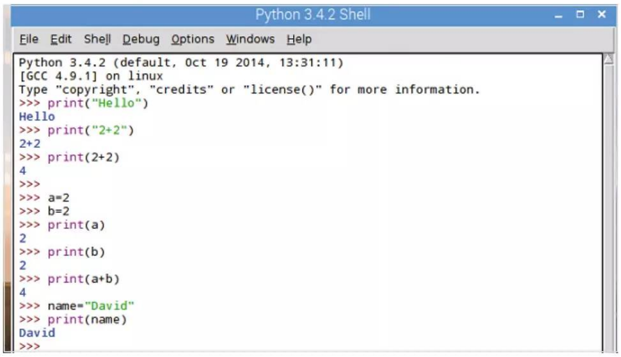


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>>
>>> a=2
>>> b=2
>>> print(a)
2
>>> print(b)
2
>>> print(a+b)
4
>>> |
```

STEP 7

You can play around with different kinds of variables and the Print function. For example, you could assign variables for someone's name:

```
name=”David”
print(name)
```



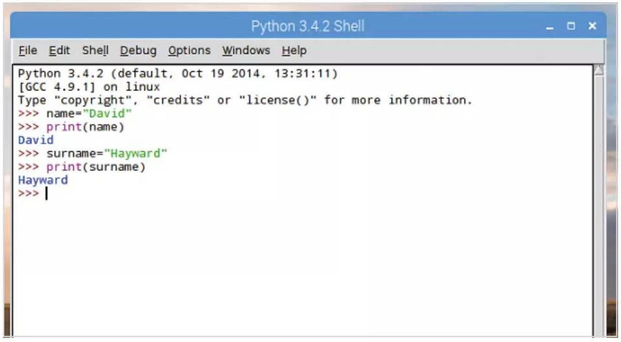
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> print("Hello")
Hello
>>> print("2+2")
2+2
>>> print(2+2)
4
>>>
>>> a=2
>>> b=2
>>> print(a)
2
>>> print(b)
2
>>> print(a+b)
4
>>> name="David"
>>> print(name)
David
>>>
```

STEP 8

Now let's add a surname:

```
surname=”Hayward”
print(surname)
```

You now have two variables containing a first name and a surname and you can print them independently.



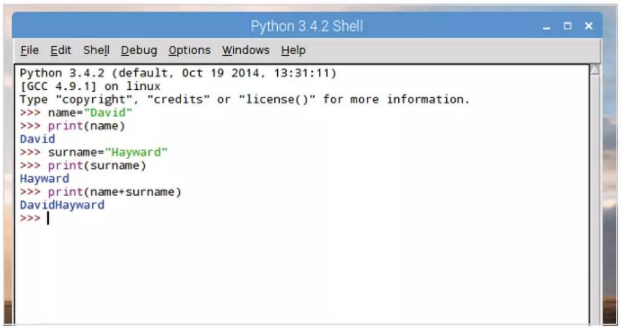
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David"
>>> print(name)
David
>>> surname="Hayward"
>>> print(surname)
Hayward
>>> |
```

STEP 9

If we were to apply the same routine as before, using the + symbol, the name wouldn't appear correctly in the output in the Shell. Try it:

```
print(name+surname)
```

You need a space between the two, defining them as two separate values and not something you mathematically play around with.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David"
>>> print(name)
David
>>> surname="Hayward"
>>> print(surname)
Hayward
>>> print(name+surname)
DavidHayward
>>> |
```

STEP 10

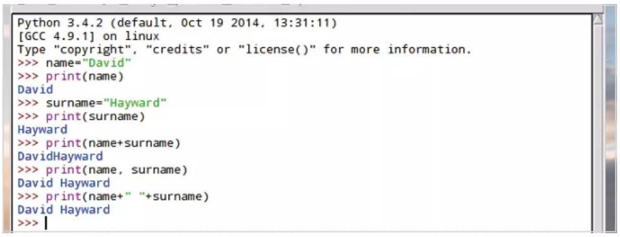
In Python 3 you can separate the two variables with a space using a comma:

```
print(name, surname)
```

Alternatively, you can add the space yourself:

```
print(name+” “+surname)
```

The use of the comma is much neater, as you can see. Congratulations, you've just taken your first steps into the wide world of Python.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David"
>>> print(name)
David
>>> surname="Hayward"
>>> print(surname)
Hayward
>>> print(name+surname)
DavidHayward
>>> print(name, surname)
David Hayward
>>> print(name+" "+surname)
David Hayward
>>> |
```

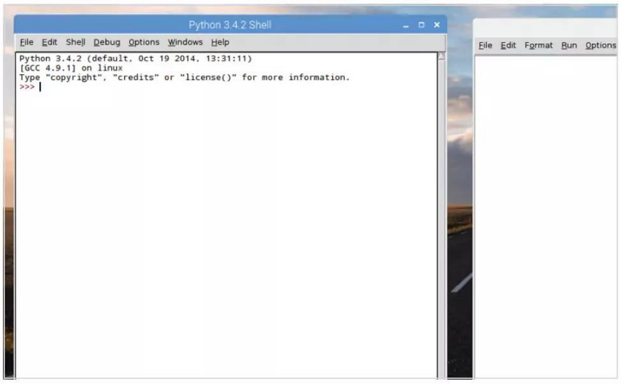
Saving and Executing Your Code

Whilst working in the IDLE Shell is perfectly fine for small code snippets, it's not designed for entering longer program listings. In this section you're going to be introduced to the IDLE Editor, where you will be working from now on.

EDITING CODE

You will eventually reach a point where you have to move on from inputting single lines of code into the Shell. Instead, the IDLE Editor will allow you to save and execute your Python code.

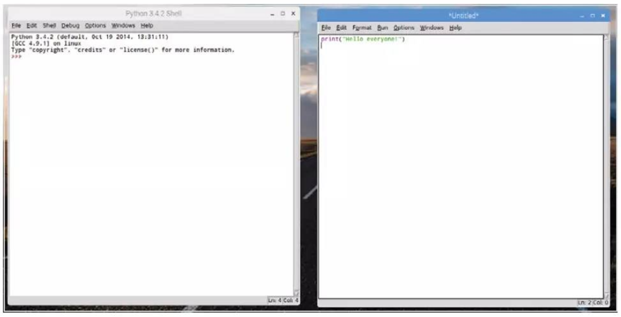
STEP 1 First, open the Python IDLE Shell and when it's up, click on File > New File. This will open a new window with Untitled as its name. This is the Python IDLE Editor and within it you can enter the code needed to create your future programs.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
File Edit Format Run Options

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> |
```

STEP 2 The IDLE Editor is, for all intents and purposes, a simple text editor with Python features, colour coding and so on; much in the same vein as Sublime. You enter code as you would within the Shell, so taking an example from the previous tutorial, enter:

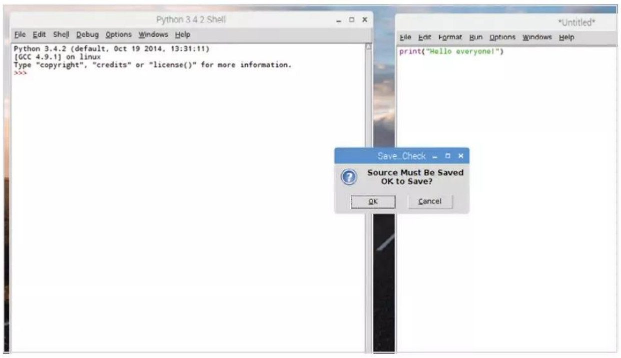
```
print("Hello everyone!")
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Untitled
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

```
print("Hello everyone!")
```

STEP 3 You can see that the same colour coding is in place in the IDLE Editor as it is in the Shell, enabling you to better understand what's going on with your code. However, to execute the code you need to first save it. Press F5 and you get a Save…Check box open.

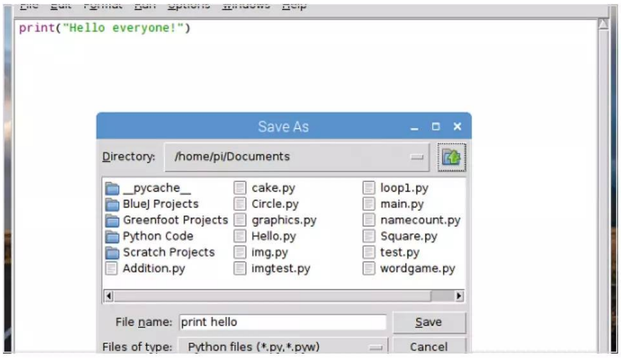

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Untitled
File Edit Format Run Options Windows Help
Save..Check
OK
Cancel

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

```
print("Hello everyone!")
```

Source Must Be Saved
OK to Save?

STEP 4 Click on the OK button in the Save box and select a destination where you'll save all your Python code. The destination can be a dedicated folder called Python or you can just dump it wherever you like. Remember to keep a tidy drive though, to help you out in the future.


File Edit Format Run Options Windows Help
Save As
Save
Cancel
__pycache__
BlueJ Projects
Greenfoot Projects
Python Code
Scratch Projects
Addition.py
cake.py
Circle.py
graphics.py
Hello.py
img.py
imgtest.py
loop1.py
main.py
namecount.py
Square.py
test.py
wordgame.py

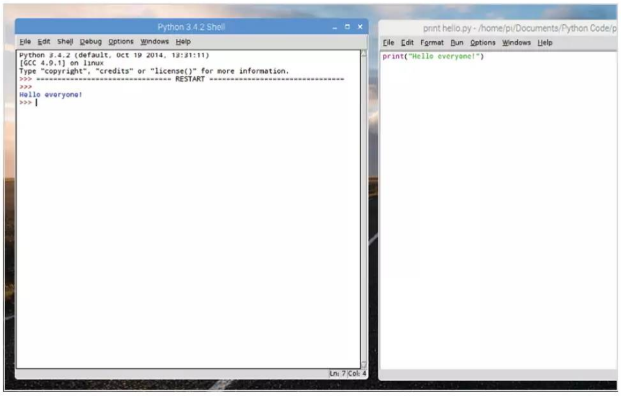
```
print("Hello everyone!")
```

Directory: /home/pi/Documents

File name: print hello

Files of type: Python files (*.py,*.pyw)

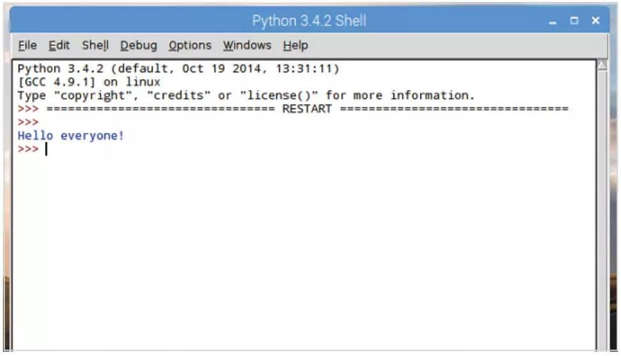
STEP 5 Enter a name for your code, 'print hello' for example, and click on the Save button. Once the Python code is saved it's executed and the output will be detailed in the IDLE Shell. In this case, the words 'Hello everyone!'.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
print hello.py - /home/pi/Documents/Python Code/p
File Edit Format Run Options Windows Help
Ln: 7 Col: 4

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello everyone!
>>> 
```

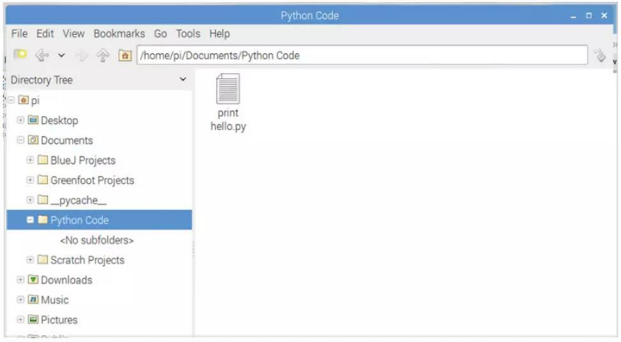
```
print("Hello everyone!")
```

STEP 6 This is how the vast majority of your Python code will be conducted. Enter it into the Editor, hit F5, save the code and look at the output in the Shell. Sometimes things will differ, depending on whether you've requested a separate window, but essentially that's the process. It's the process we will use throughout this book, unless otherwise stated.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello everyone!
>>> 
```

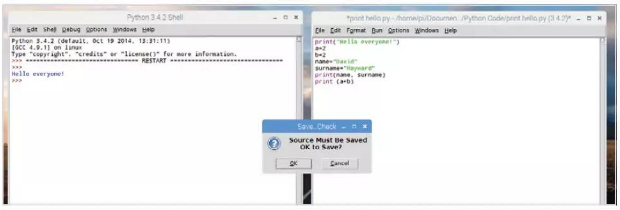
STEP 7 If you open the file location of the saved Python code, you can see that it ends in a .py extension. This is the default Python file name. Any code you create will be whatever.py and any code downloaded from the many Internet Python resource sites will be .py. Just ensure that the code is written for Python 3.


Python Code
File Edit View Bookmarks Go Tools Help
/home/pi/Documents/Python Code
Directory Tree
pi
Desktop
Documents
BlueJ Projects
Greenfoot Projects
__pycache__
Python Code
<No subfolders>
Scratch Projects
Downloads
Music
Pictures
print
hello.py

STEP 8 Let's extend the code and enter a few examples from the previous tutorial:

```
a=2
b=2
name=”David”
surname=”Hayward”
print(name, surname)
print (a+b)
```

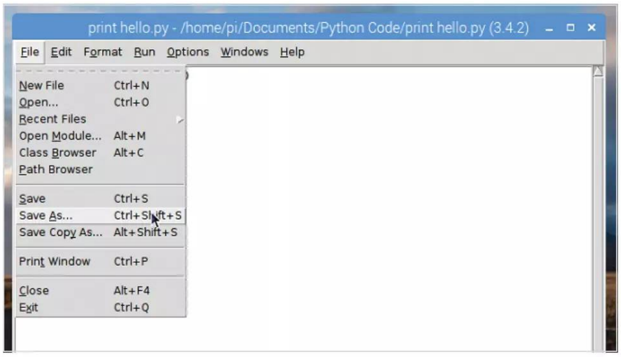
If you press F5 now you'll be asked to save the file, again, as it's been modified from before.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
print hello.py - /home/pi/Documen../Python Code/print hello.py (3.4.2)
File Edit Format Run Options Windows Help
Save..Check
Source Must Be Saved
OK to Save?
OK
Cancel

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello everyone!
>>>
```

```
print("Hello everyone!")
a=2
b=2
name="David"
surname="Hayward"
print(name, surname)
print (a+b)
```

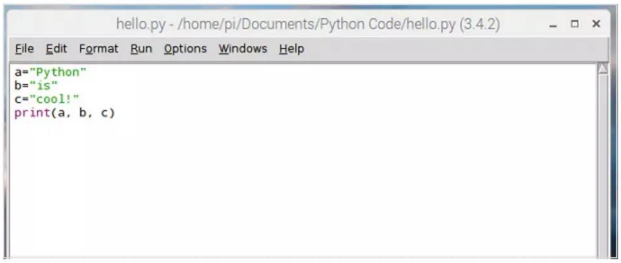
STEP 9 If you click the OK button, the file will be overwritten with the new code entries, and executed, with the output in the Shell. It's not a problem with just these few lines but if you were to edit a larger file, overwriting can become an issue. Instead, use File > Save As from within the Editor to create a backup.


print hello.py - /home/pi/Documents/Python Code/print hello.py (3.4.2)
File Edit Format Run Options Windows Help
New File Ctrl+N
Open... Ctrl+O
Recent Files
Open Module... Alt+M
Class Browser Alt+C
Path Browser
Save Ctrl+S
Save As... Ctrl+Shift+S
Save Copy As... Alt+Shift+S
Print Window Ctrl+P
Close Alt+F4
Exit Ctrl+Q

STEP 10 Now create a new file. Close the Editor, and open a new instance (File > New File from the Shell). Enter the following and save it as hello.py:

```
a=”Python”
b=”is”
c=”cool!”
print(a, b, c)
```

You will use this code in the next tutorial.


hello.py - /home/pi/Documents/Python Code/hello.py (3.4.2)
File Edit Format Run Options Windows Help

```
a="Python"
b="is"
c="cool!"
print(a, b, c)
```

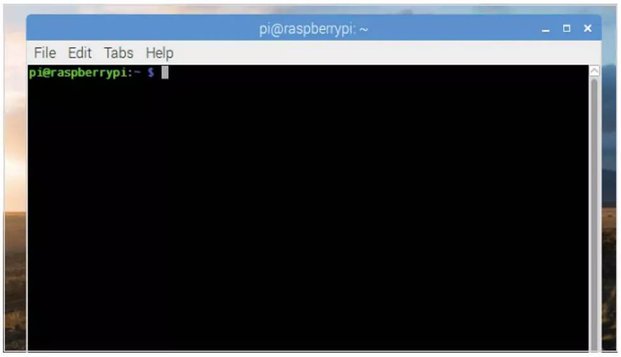
Executing Code from the Command Line

Although we're working from the GUI IDLE throughout this book, it's worth taking a look at Python's command line handling. We already know there's a command line version of Python but it's also used to execute code.

COMMAND THE CODE

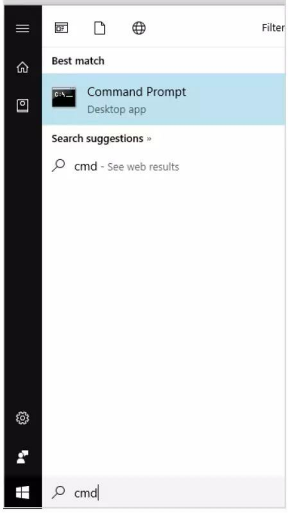
Using the code we created in the previous tutorial, the one we named hello.py, let's see how you can run code that was made in the GUI at the command line level.

STEP 1 Python, in Linux, comes with two possible ways of executing code via the command line. One of the ways is with Python 2, whilst the other uses the Python 3 libraries and so on. First though, drop into the command line or Terminal on your operating system.

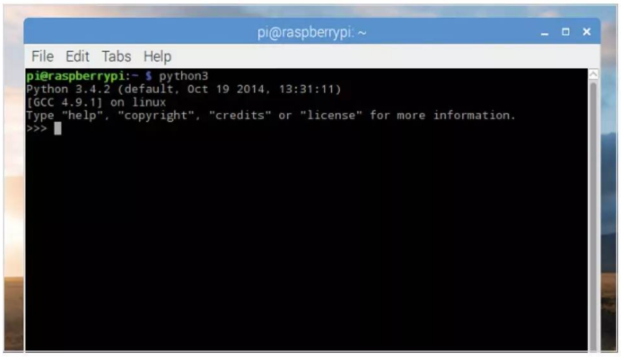

pi@raspberrypi: ~
File Edit Tabs Help

```
pi@raspberrypi:~ $ 
```

STEP 2 Just as before, we're using a Raspberry Pi: Windows users will need to click the Start button and search for CMD, then click the Command Line returned search; and macOS users can get access to their command line by clicking Go > Utilities > Terminal.


Filter
Best match
Command Prompt
Desktop app
Search suggestions
cmd - See web results
cmd

STEP 3 Now you're at the command line we can start Python. For Python 3 you need to enter the command **python3** and press Enter. This will put you into the command line version of the Shell, with the familiar three right-facing arrows as the cursor (>>>).

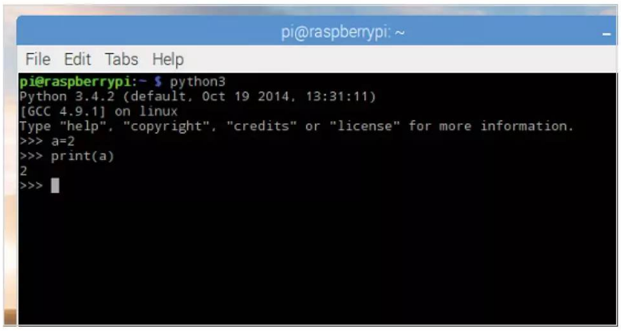

pi@raspberrypi: ~
File Edit Tabs Help

```
pi@raspberrypi:~ $ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

STEP 4 From here you're able to enter the code you've looked at previously, such as:

```
a=2
print(a)
```

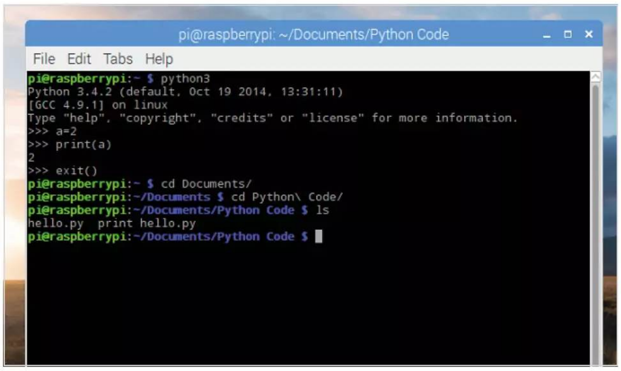
You can see that it works exactly the same.


pi@raspberrypi: ~
File Edit Tabs Help

```
pi@raspberrypi:~ $ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a=2
>>> print(a)
2
>>> 
```

STEP 5

Now enter: **exit()** to leave the command line Python session and return you back to the command prompt. Enter the folder where you saved the code from the previous tutorial and list the available files within; hopefully you should see the hello.py file.


pi@raspberrypi: ~/Documents/Python Code
File Edit Tabs Help

```
pi@raspberrypi:~ $ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a=2
>>> print(a)
2
>>> exit()
pi@raspberrypi:~ $ cd Documents/
pi@raspberrypi:~/Documents $ cd Python\ Code/
pi@raspberrypi:~/Documents/Python Code $ ls
hello.py  print hello.py
pi@raspberrypi:~/Documents/Python Code $ 
```

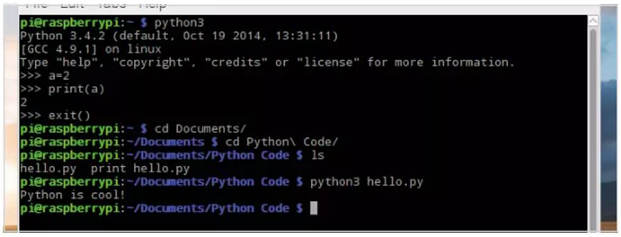
STEP 6

From within the same folder as the code you're going to run, enter the following into the command line:

```
python3 hello.py
```

This will execute the code we created, which to remind you is:

```
a="Python"
b="is"
c="cool!"
print(a, b, c)
```

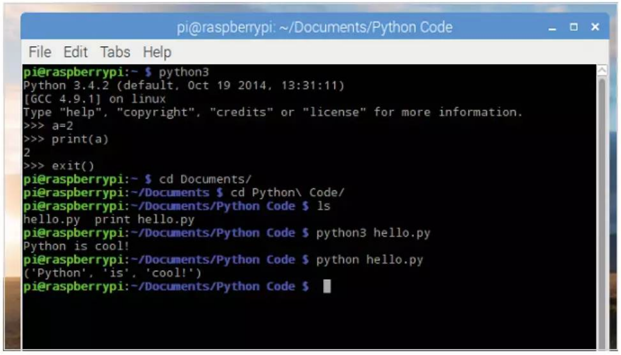


```
pi@raspberrypi:~ $ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a=2
>>> print(a)
2
>>> exit()
pi@raspberrypi:~ $ cd Documents/
pi@raspberrypi:~/Documents $ cd Python\ Code/
pi@raspberrypi:~/Documents/Python Code $ ls
hello.py  print hello.py
pi@raspberrypi:~/Documents/Python Code $ python3 hello.py
Python is cool!
pi@raspberrypi:~/Documents/Python Code $ 
```

STEP 7

Naturally, since this is Python 3 code, using the syntax and layout that's unique to Python 3, it only works when you use the python3 command. If you like, try the same with Python 2 by entering:

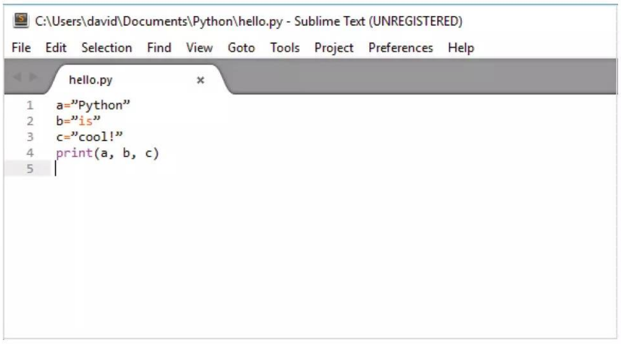
```
python hello.py
```


pi@raspberrypi: ~/Documents/Python Code
File Edit Tabs Help

```
pi@raspberrypi:~ $ python3
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a=2
>>> print(a)
2
>>> exit()
pi@raspberrypi:~ $ cd Documents/
pi@raspberrypi:~/Documents $ cd Python\ Code/
pi@raspberrypi:~/Documents/Python Code $ ls
hello.py  print hello.py
pi@raspberrypi:~/Documents/Python Code $ python3 hello.py
Python is cool!
pi@raspberrypi:~/Documents/Python Code $ python hello.py
('Python', 'is', 'cool!')
pi@raspberrypi:~/Documents/Python Code $ 
```

STEP 8

The result of running Python 3 code from the Python 2 command line is quite obvious. Whilst it doesn't error out in any way, due to the differences between the way Python 3 handles the Print command over Python 2, the result isn't as we expected. Using Sublime for the moment, open the hello.py file.

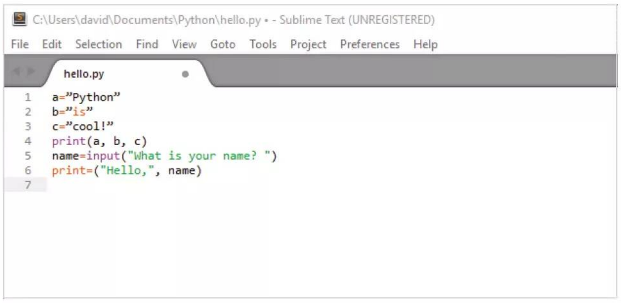

C:\Users\david\Documents\Python\hello.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
hello.py

```
a="Python"
b="is"
c="cool!"
print(a, b, c)
```

STEP 9

Since Sublime Text isn't available for the Raspberry Pi, you're going to temporarily leave the Pi for the moment and use Sublime as an example that you don't necessarily need to use the Python IDLE. With the hello.py file open, alter it to include the following:

```
name=input("What is your name? ")
print("Hello,", name)
```


C:\Users\david\Documents\Python\hello.py • - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
hello.py

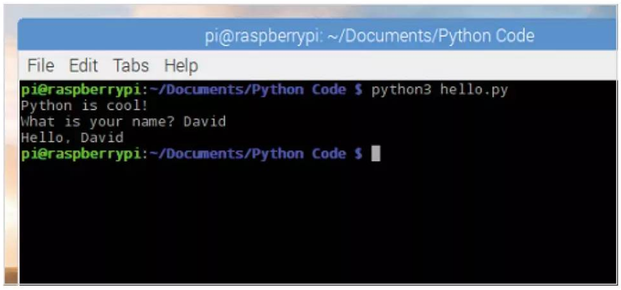
```
a="Python"
b="is"
c="cool!"
print(a, b, c)
name=input("What is your name? ")
print=("Hello,", name)
```

STEP 10

Save the hello.py file and drop back to the command line. Now execute the newly saved code with:

```
python3 hello.py
```

The result will be the original Python is cool! statement, together with the added input command asking you for your name, and displaying it in the command window.


pi@raspberrypi: ~/Documents/Python Code
File Edit Tabs Help

```
pi@raspberrypi:~/Documents/Python Code $ python3 hello.py
Python is cool!
What is your name? David
Hello, David
pi@raspberrypi:~/Documents/Python Code $ 
```

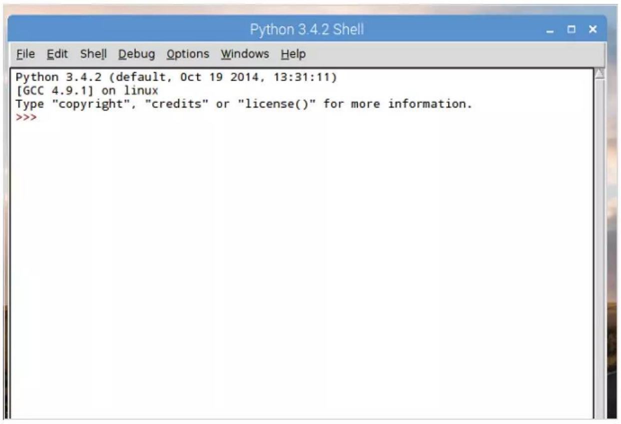
Numbers and Expressions

We've seen some basic mathematical expressions with Python, simple addition and the like. Let's expand on that now and see just how powerful Python is as a calculator. You can work within the IDLE Shell or in the Editor, whichever you like.

IT'S ALL MATHS, MAN

You can get some really impressive results with the mathematical powers of Python; as with most, if not all, programming languages, maths is the driving force behind the code.

STEP 1 Open up the GUI version of Python 3, as mentioned you can use either the Shell or the Editor. For the time being, you're going to use the Shell just to warm our maths muscle, which we believe is a small gland located at the back of the brain (or not).

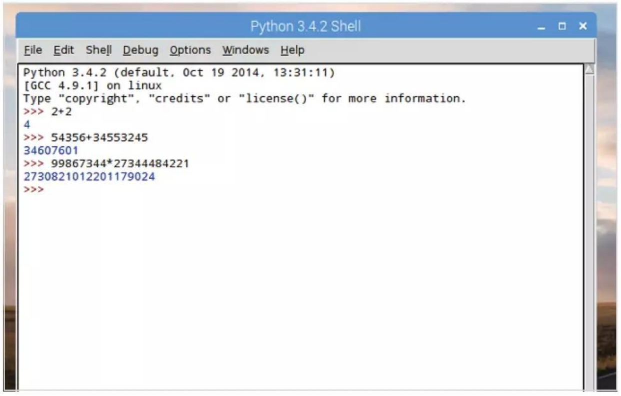

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

STEP 2 In the Shell enter the following:

```
2+2
54356+34553245
99867344*27344484221
```

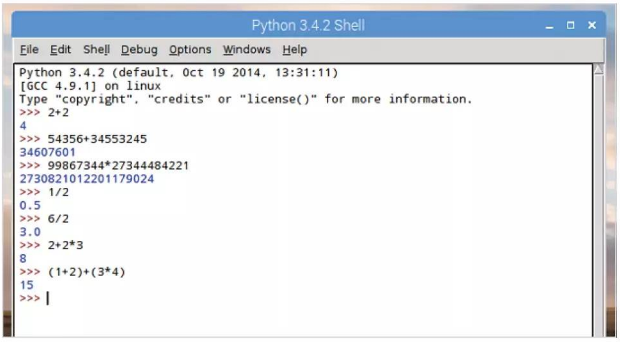
You can see that Python can handle some quite large numbers.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>>
```

STEP 3 You can use all the usual mathematical operations: divide, multiply, brackets and so on. Practise with a few, for example:

```
1/2
6/2
2+2*3
(1+2)+(3*4)
```

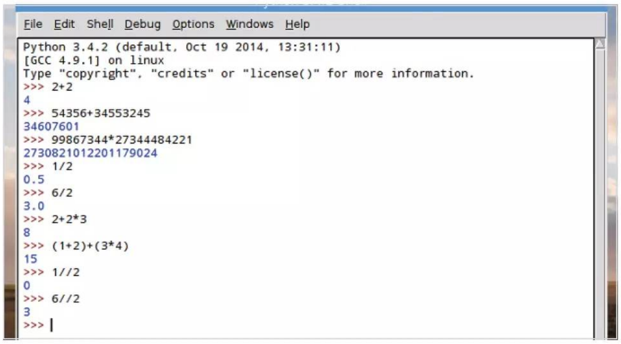

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> |
```

STEP 4 You've no doubt noticed, division produces a decimal number. In Python these are called floats, or floating point arithmetic. However, if you need an integer as opposed to a decimal answer, then you can use a double slash:

```
1//2
6//2
```

And so on.


File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> 2+2
4
>>> 54356+34553245
34607601
>>> 99867344*27344484221
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> |
```

STEP 5

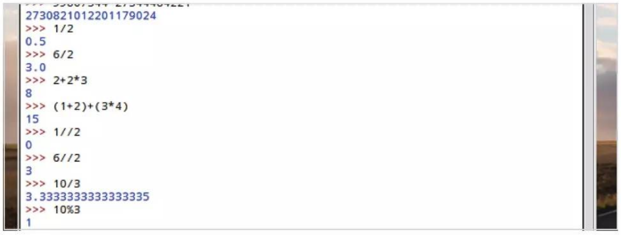
You can also use an operation to see the remainder left over from division. For example:

```
10/3
```

Will display 3.333333333, which is of course 3.3-recurring. If you now enter:

```
10%3
```

This will display 1, which is the remainder left over from dividing 10 into 3.



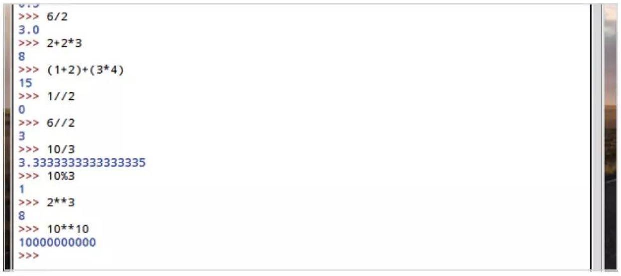
```
2730821012201179024
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
```

STEP 6

Next up we have the power operator, or exponentiation if you want to be technical. To work out the power of something you can use a double multiplication symbol or double-star on the keyboard:

```
2**3
10**10
```

Essentially, it's 2x2x2 but we're sure you already know the basics behind maths operators. This is how you would work it out in Python.

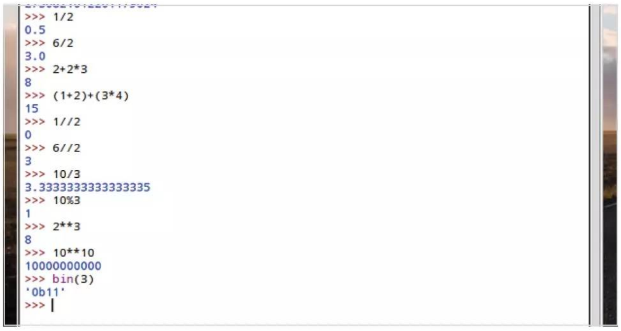


```
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>>
```

STEP 7

Numbers and expressions don't stop there. Python has numerous built-in functions to work out sets of numbers, absolute values, complex numbers and a host of mathematical expressions and Pythagorean tongue-twisters. For example, to convert a number to binary, use:

```
bin(3)
```



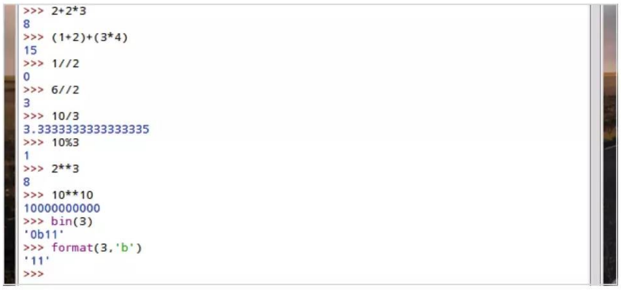
```
>>> 1/2
0.5
>>> 6/2
3.0
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>> bin(3)
'0b11'
>>> |
```

STEP 8

This will be displayed as '0b11', converting the integer into binary and adding the prefix 0b to the front. If you want to remove the 0b prefix, then you can use:

```
format(3, 'b')
```

The Format command converts a value, the number 3, to a formatted representation as controlled by the format specification, the 'b' part.

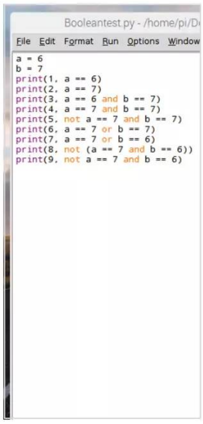


```
>>> 2+2*3
8
>>> (1+2)+(3*4)
15
>>> 1//2
0
>>> 6//2
3
>>> 10/3
3.3333333333333335
>>> 10%3
1
>>> 2**3
8
>>> 10**10
10000000000
>>> bin(3)
'0b11'
>>> format(3,'b')
'11'
>>>
```

STEP 9

A Boolean Expression is a logical statement that will either be true or false. We can use these to compare data and test to see if it's equal to, less than or greater than. Try this in a New File:

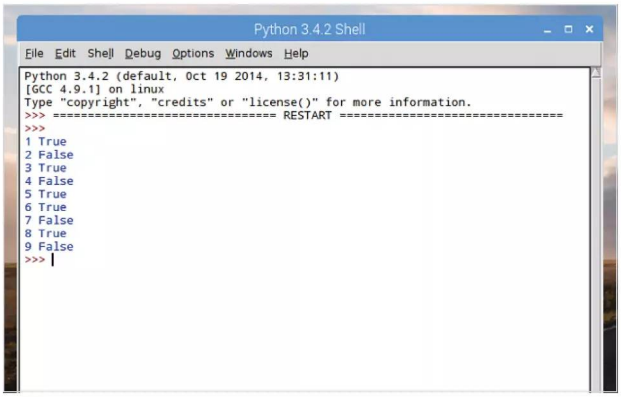
```
a = 6
b = 7
print(1, a == 6)
print(2, a == 7)
print(3, a == 6 and b == 7)
print(4, a == 7 and b == 7)
print(5, not a == 7 and b == 7)
print(6, a == 7 or b == 7)
print(7, a == 7 or b == 6)
print(8, not (a == 7 and b == 6))
print(9, not a == 7 and b == 6)
```


Booleantest.py - /home/pi/D
File Edit Format Run Options Window

```
a = 6
b = 7
print(1, a == 6)
print(2, a == 7)
print(3, a == 6 and b == 7)
print(4, a == 7 and b == 7)
print(5, not a == 7 and b == 7)
print(6, a == 7 or b == 7)
print(7, a == 7 or b == 6)
print(8, not (a == 7 and b == 6))
print(9, not a == 7 and b == 6)
```

STEP 10

Execute the code from Step 9, and you can see a series of True or False statements, depending on the result of the two defining values: 6 and 7. It's an extension of what you've looked at, and an important part of programming.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
1 True
2 False
3 True
4 False
5 True
6 True
7 False
8 True
9 False
>>> |
```

Using Comments

When writing your code, the flow of it, what each variable does, how the overall program will operate and so on is all inside your head. Another programmer could follow the code line by line but over time, it can become difficult to read.

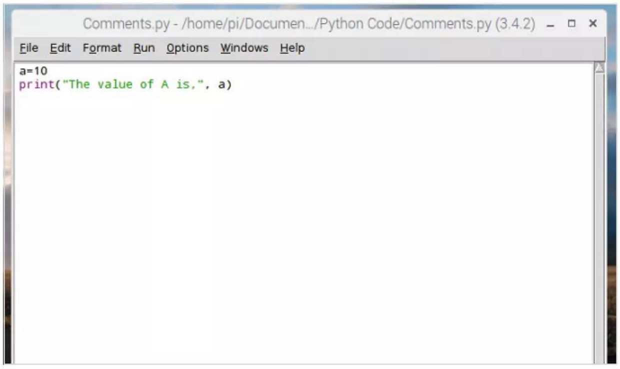
#COMMENTS!

Programmers use a method of keeping their code readable by commenting on certain sections. If a variable is used, the programmer comments on what it's supposed to do, for example. It's just good practise.

STEP 1 Start by creating a new instance of the IDLE Editor (File > New File) and create a simple variable and print command:

```
a=10
print("The value of A is,", a)
```

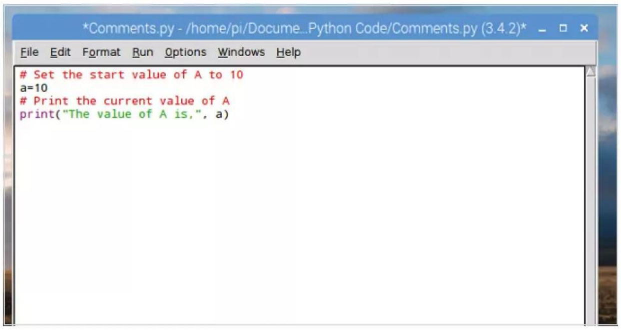
Save the file and execute the code.


Comments.py - /home/pi/Documen.../Python Code/Comments.py (3.4.2)
File Edit Format Run Options Windows Help

```
a=10
print("The value of A is,", a)
```

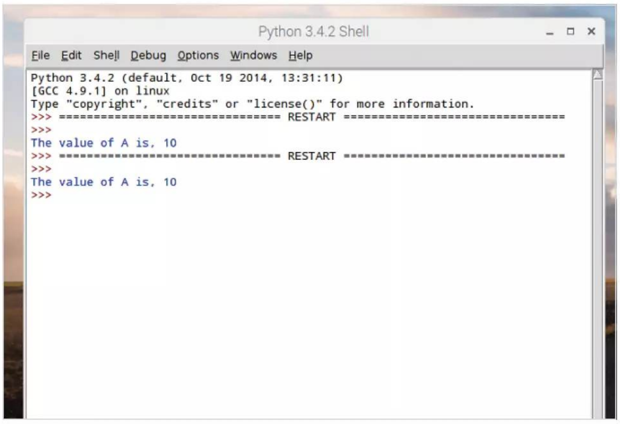
STEP 2 Running the code will return the line: The value of A is, 10 into the IDLE Shell window, which is what we expected. Now, add some of the types of comments you'd normally see within code:

```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
```


Comments.py - /home/pi/Docume...Python Code/Comments.py (3.4.2)
File Edit Format Run Options Windows Help

```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
```

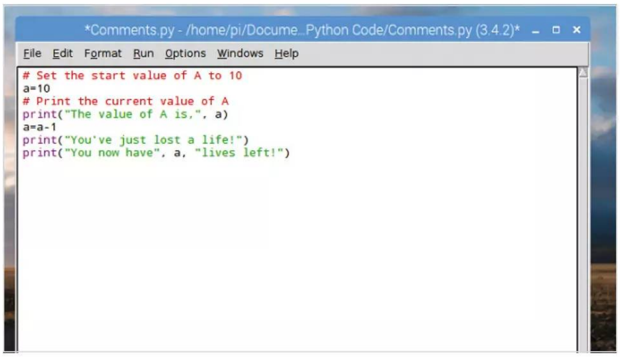
STEP 3 Resave the code and execute it. You can see that the output in the IDLE Shell is still the same as before, despite the extra lines being added. Simply put, the hash symbol (#) denotes a line of text the programmer can insert to inform them, and others, of what's going on without the user being aware.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> =============================== RESTART ===============================
>>>
The value of A is, 10
>>> =============================== RESTART ===============================
>>>
The value of A is, 10
>>>
```

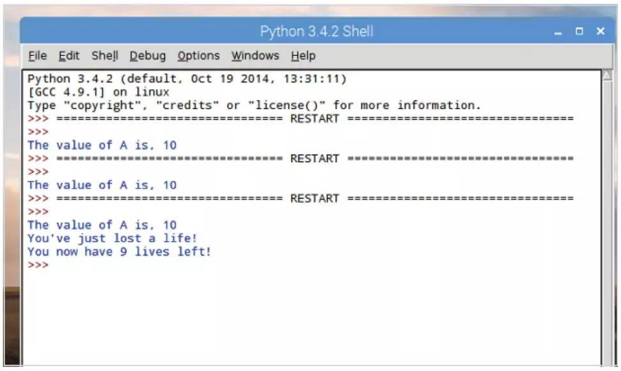
STEP 4 Let's assume that the variable A that we've created is the number of lives in a game. Every time the player dies, the value is decreased by 1. The programmer could insert a routine along the lines of:

```
a=a-1
print("You've just lost a life!")
print("You now have", a, "lives left!")
```


Comments.py - /home/pi/Docume...Python Code/Comments.py (3.4.2)
File Edit Format Run Options Windows Help

```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
a=a-1
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

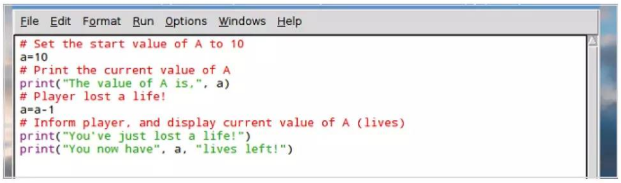
STEP 5 Whilst we know that the variable A is lives, and that the player has just lost one, a casual viewer or someone checking the code may not know. Imagine for a moment that the code is twenty thousand lines long, instead of just our seven. You can see how handy comments are.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
The value of A is, 10
>>> ================================ RESTART ================================
>>>
The value of A is, 10
>>> ================================ RESTART ================================
>>>
The value of A is, 10
You've just lost a life!
You now have 9 lives left!
>>>
```

STEP 6 Essentially, the new code together with comments could look like:

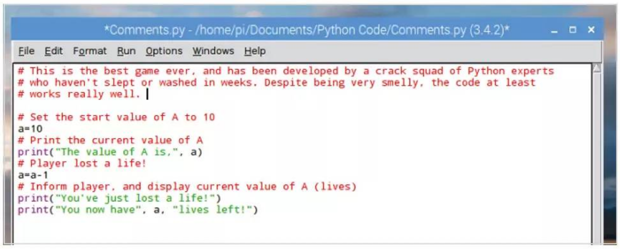
```
# Set the start value of A to 10
a=10
# Print the current value of A
print(“The value of A is,”, a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A
(lives)
print(“You’ve just lost a life!”)
print(“You now have”, a, “lives left!”)
```


File Edit Format Run Options Windows Help

```
# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

STEP 7 You can use comments in different ways. For example, Block Comments are a large section of text that details what’s going on in the code, such as telling the code reader what variables you’re planning on using:

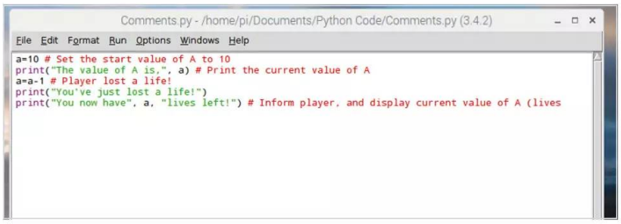
```
# This is the best game ever, and has been
developed by a crack squad of Python experts
# who haven’t slept or washed in weeks. Despite
being very smelly, the code at least
# works really well.
```


Comments.py - /home/pi/Documents/Python Code/Comments.py (3.4.2)
File Edit Format Run Options Windows Help

```
# This is the best game ever, and has been developed by a crack squad of Python experts
# who haven't slept or washed in weeks. Despite being very smelly, the code at least
# works really well.

# Set the start value of A to 10
a=10
# Print the current value of A
print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

STEP 8 Inline comments are comments that follow a section of code. Take our examples from above, instead of inserting the code on a separate line, we could use:

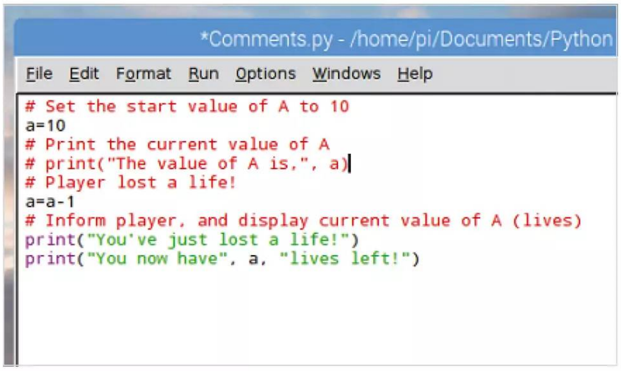
```
a=10 # Set the start value of A to 10
print(“The value of A is,”, a) # Print the current
value of A
a=a-1 # Player lost a life!
print(“You’ve just lost a life!”)
print(“You now have”, a, “lives left!”) # Inform
player, and display current value of A (lives)
```


Comments.py - /home/pi/Documents/Python Code/Comments.py (3.4.2)
File Edit Format Run Options Windows Help

```
a=10 # Set the start value of A to 10
print("The value of A is,", a) # Print the current value of A
a=a-1 # Player lost a life!
print("You've just lost a life!")
print("You now have", a, "lives left!") # Inform player, and display current value of A (lives
```

STEP 9 The comment, the hash symbol, can also be used to comment out sections of code you don’t want to be executed in your program. For instance, if you wanted to remove the first print statement, you would use:

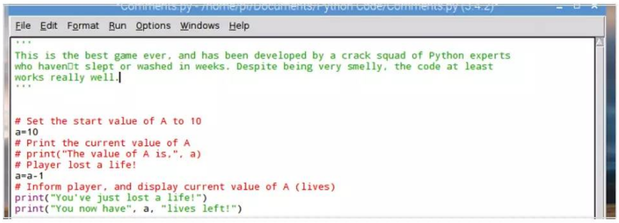
```
# print(“The value of A is,”, a)
```


*Comments.py - /home/pi/Documents/Python
File Edit Format Run Options Windows Help

```
# Set the start value of A to 10
a=10
# Print the current value of A
# print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

STEP 10 You also use three single quotes to comment out a Block Comment or multi-line section of comments. Place them before and after the areas you want to comment for them to work:

```
‘’’
This is the best game ever, and has been developed
by a crack squad of Python experts who haven’t
slept or washed in weeks. Despite being very
smelly, the code at least works really well.
’’’
```


File Edit Format Run Options Windows Help

```
'''
This is the best game ever, and has been developed by a crack squad of Python experts
who haven□t slept or washed in weeks. Despite being very smelly, the code at least
works really well.
'''


# Set the start value of A to 10
a=10
# Print the current value of A
# print("The value of A is,", a)
# Player lost a life!
a=a-1
# Inform player, and display current value of A (lives)
print("You've just lost a life!")
print("You now have", a, "lives left!")
```

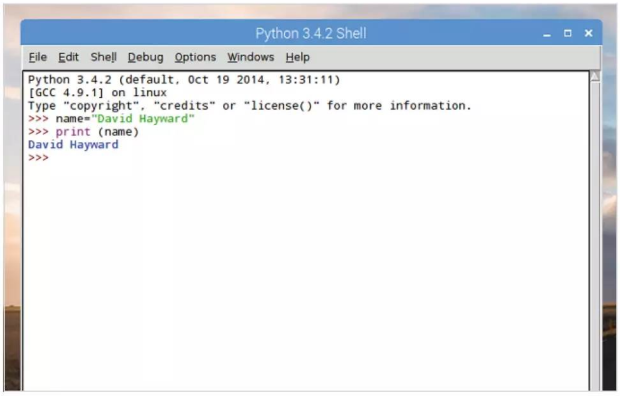
Working with Variables

We've seen some examples of variables in our Python code already but it's always worth going through the way they operate and how Python creates and assigns certain values to a variable.

VARIOUS VARIABLES

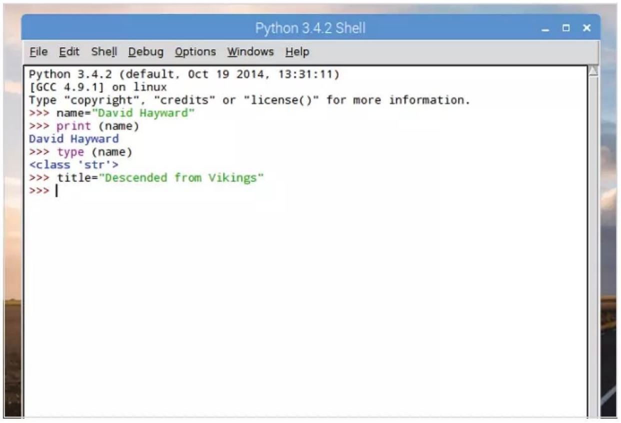
You'll be working with the Python 3 IDLE Shell in this tutorial. If you haven't already, open Python 3 or close down the previous IDLE Shell to clear up any old code.

STEP 1 In some programming languages you're required to use a dollar sign to denote a string, which is a variable made up of multiple characters, such as a name of a person. In Python this isn't necessary. For example, in the Shell enter: `name=”David Hayward”` (or use your own name, unless you're also called David Hayward).



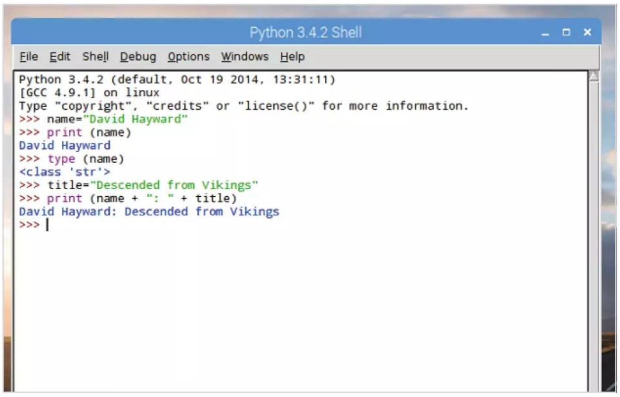
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print (name)
David Hayward
>>>
```

STEP 2 You can check the type of variable in use by issuing the **type ()** command, placing the name of the variable inside the brackets. In our example, this would be: `type (name)`. Add a new string variable: `title=”Descended from Vikings”`.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print (name)
David Hayward
>>> type (name)
<class 'str'>
>>> title="Descended from Vikings"
>>> |
```

STEP 3 You've seen previously that variables can be concatenated using the plus symbol between the variable names. In our example we can use: `print (name + “: “ + title)`. The middle part between the quotations allows us to add a colon and a space, as variables are connected without spaces, so we need to add them manually.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print (name)
David Hayward
>>> type (name)
<class 'str'>
>>> title="Descended from Vikings"
>>> print (name + ": " + title)
David Hayward: Descended from Vikings
>>> |
```

STEP 4 You can also combine variables within another variable. For example, to combine both name and title variables into a new variable we use:

```
character=name + “: “ + title
```

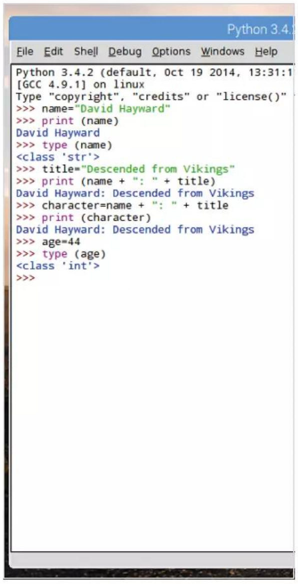
Then output the content of the new variable as:

```
print (character)
```

Numbers are stored as different variables:

```
age=44
Type (age)
```

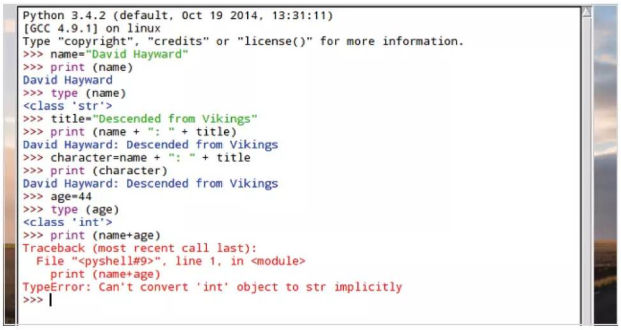
Which, as we know, are integers.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:1
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()"
>>> name="David Hayward"
>>> print (name)
David Hayward
>>> type (name)
<class 'str'>
>>> title="Descended from Vikings"
>>> print (name + ": " + title)
David Hayward: Descended from Vikings
>>> character=name + ": " + title
>>> print (character)
David Hayward: Descended from Vikings
>>> age=44
>>> type (age)
<class 'int'>
>>>
```

STEP 5 However, you can't combine both strings and integer type variables in the same command, as you would a set of similar variables. You need to either turn one into the other or vice versa. When you do try to combine both, you get an error message:

```
print (name + age)
```



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> name="David Hayward"
>>> print (name)
David Hayward
>>> type (name)
<class 'str'>
>>> title="Descended from Vikings"
>>> print (name + ": " + title)
David Hayward: Descended from Vikings
>>> character=name + ": " + title
>>> print (character)
David Hayward: Descended from Vikings
>>> age=44
>>> type (age)
<class 'int'>
>>> print (name+age)
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    print (name+age)
TypeError: Can't convert 'int' object to str implicitly
>>> |
```

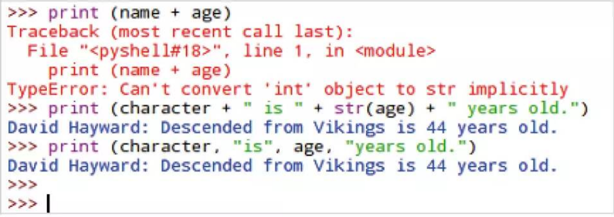
STEP 6 This is a process known as TypeCasting. The Python code is:

```
print (character + " is " + str(age) + " years
old.")
```

or you can use:

```
print (character, "is", age, "years old.")
```

Notice again that in the last example, you don't need the spaces between the words in quotes as the commas treat each argument to print separately.

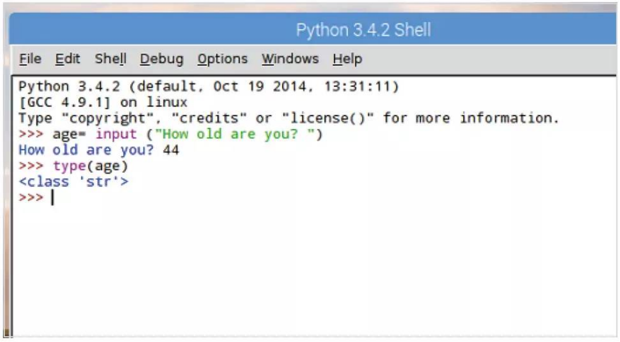


```
>>> print (name + age)
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    print (name + age)
TypeError: Can't convert 'int' object to str implicitly
>>> print (character + " is " + str(age) + " years old.")
David Hayward: Descended from Vikings is 44 years old.
>>> print (character, "is", age, "years old.")
David Hayward: Descended from Vikings is 44 years old.
>>>
>>> |
```

STEP 7 Another example of TypeCasting is when you ask for input from the user, such as a name. for example, enter:

```
age= input ("How old are you? ")
```

All data stored from the Input command is stored as a string variable.

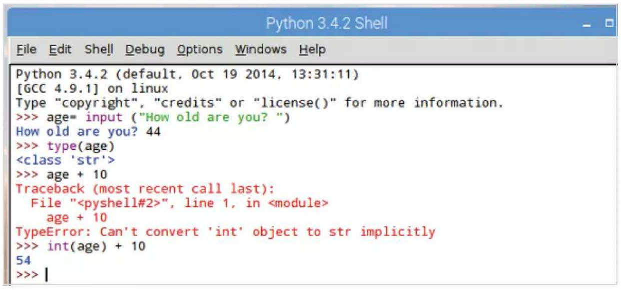


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> age= input ("How old are you? ")
How old are you? 44
>>> type(age)
<class 'str'>
>>> |
```

STEP 8 This presents a bit of a problem when you want to work with a number that's been inputted by the user, as age + 10 won't work due to being a string variable and an integer. Instead, you need to enter:

```
int(age) + 10
```

This will TypeCast the age string into an integer that can be worked with.

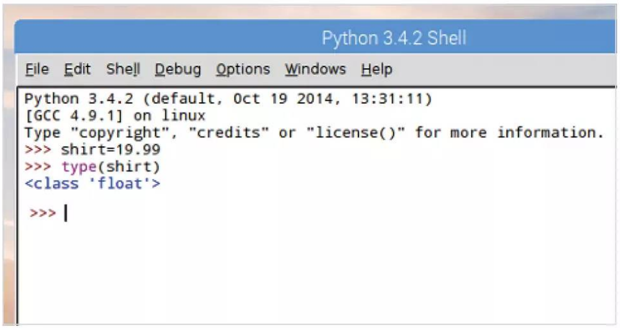


```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> age= input ("How old are you? ")
How old are you? 44
>>> type(age)
<class 'str'>
>>> age + 10
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    age + 10
TypeError: Can't convert 'int' object to str implicitly
>>> int(age) + 10
54
>>> |
```

STEP 9 The use of TypeCasting is also important when dealing with floating point arithmetic; remember: numbers that have a decimal point in them. For example, enter:

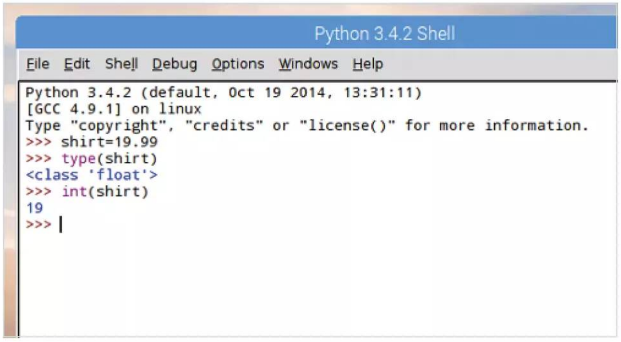
```
shirt=19.99
```

Now enter `type(shirt)` and you'll see that Python has allocated the number as a 'float', because the value contains a decimal point.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> shirt=19.99
>>> type(shirt)
<class 'float'>

 >>> |
```

STEP 10 When combining integers and floats Python usually converts the integer to a float, but should the reverse ever be applied it's worth remembering that Python doesn't return the exact value. When converting a float to an integer, Python will always round down to the nearest integer, called truncating; in our case instead of 19.99 it becomes 19.



```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> shirt=19.99
>>> type(shirt)
<class 'float'>
>>> int(shirt)
19
>>> |
```

User Input

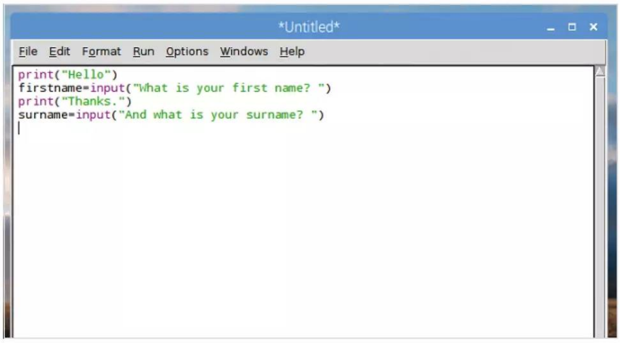
We've seen some basic user interaction with the code from a few of the examples earlier, so now would be a good time to focus solely on how you would get information from the user then store and present it.

USER FRIENDLY

The type of input you want from the user will depend greatly on the type of program you're coding. For example, a game may ask for a character's name, whereas a database can ask for personal details.

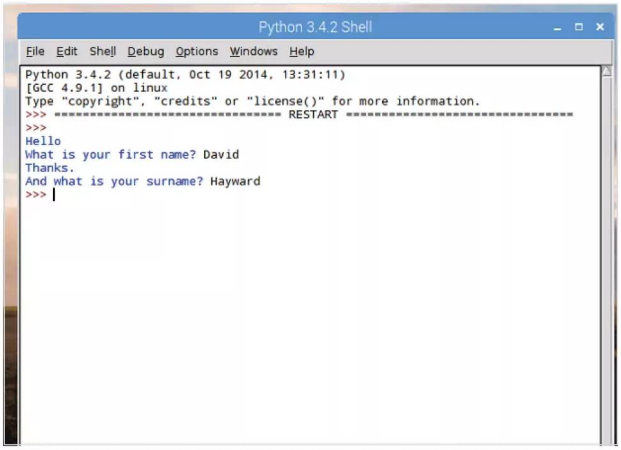
STEP 1 If it's not already, open the Python 3 IDLE Shell, and start a New File in the Editor. Let's begin with something really simple, enter:

```
print(“Hello”)
firstname=input(“What is your first name? “)
print(“Thanks.”)
surname=input(“And what is your surname? “)
```


Untitled
File Edit Format Run Options Windows Help

```
print("Hello")
firstname=input("What is your first name? ")
print("Thanks.")
surname=input("And what is your surname? ")
```

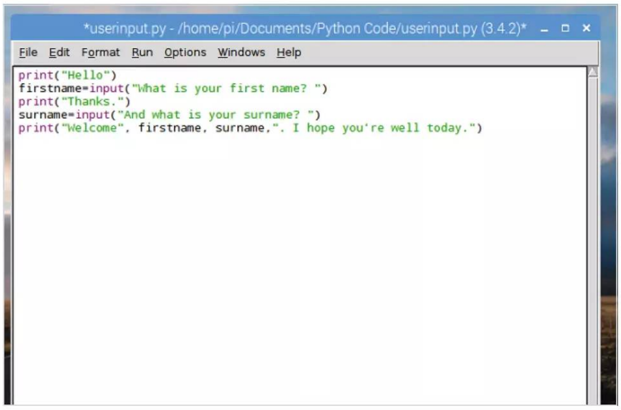
STEP 2 Save and execute the code, and as you already no doubt suspected, in the IDLE Shell the program will ask for your first name, storing it as the variable firstname, followed by your surname; also stored in its own variable (surname).


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello
What is your first name? David
Thanks.
And what is your surname? Hayward
>>> 
```

STEP 3 Now that we have the user's name stored in a couple of variables we can call them up whenever we want:

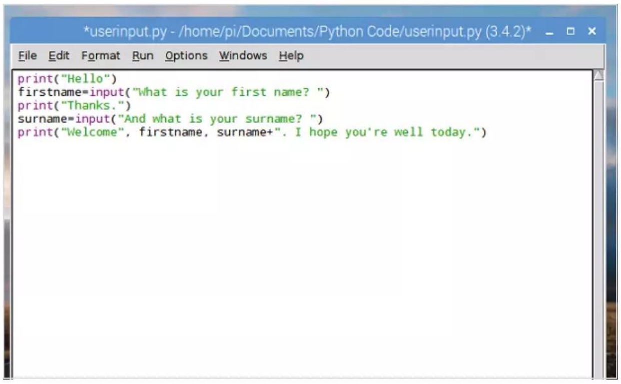
```
print(“Welcome”, firstname, surname, “. I hope
you’re well today.”)
```


userinput.py - /home/pi/Documents/Python Code/userinput.py (3.4.2)
File Edit Format Run Options Windows Help

```
print("Hello")
firstname=input("What is your first name? ")
print("Thanks.")
surname=input("And what is your surname? ")
print("Welcome", firstname, surname,". I hope you're well today.")
```

STEP 4 Run the code and you can see a slight issue, the full stop after the surname follows a blank space. To eliminate that we can add a plus sign instead of the comma in the code:

```
print(“Welcome”, firstname, surname+“. I hope
you’re well today.”)
```

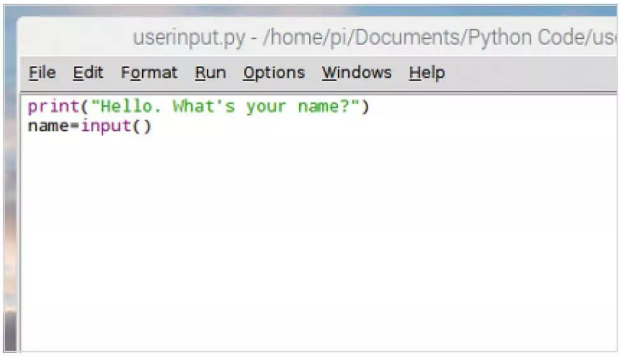

userinput.py - /home/pi/Documents/Python Code/userinput.py (3.4.2)
File Edit Format Run Options Windows Help

```
print("Hello")
firstname=input("What is your first name? ")
print("Thanks.")
surname=input("And what is your surname? ")
print("Welcome", firstname, surname+". I hope you're well today.")
```

STEP 5

You don't always have to include quoted text within the input command. For example, you can ask the user their name, and have the input in the line below:

```
print("Hello. What's your name?")
name=input()
```

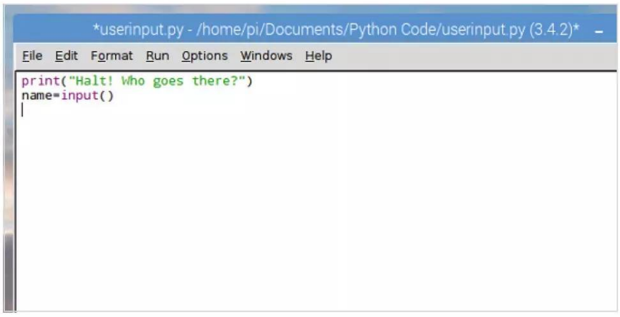

userinput.py - /home/pi/Documents/Python Code/us
File Edit Format Run Options Windows Help

```
print("Hello. What's your name?")
name=input()
```

STEP 6

The code from the previous step is often regarded as being a little neater than having a lengthy amount of text in the input command, but it's not a rule that's set in stone, so do as you like in these situations. Expanding on the code, try this:

```
print("Halt! Who goes there?")
name=input()
```

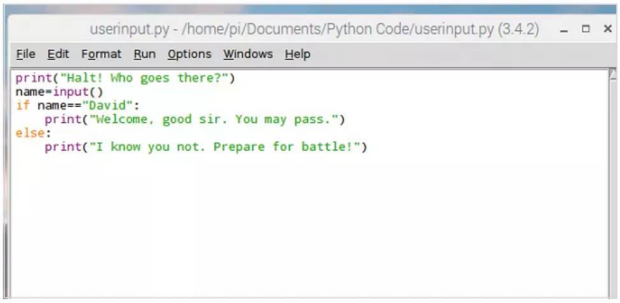

userinput.py - /home/pi/Documents/Python Code/userinput.py (3.4.2)
File Edit Format Run Options Windows Help

```
print("Halt! Who goes there?")
name=input()
```

STEP 7

It's a good start to a text adventure game, perhaps? Now you can expand on it and use the raw input from the user to flesh out the game a little:

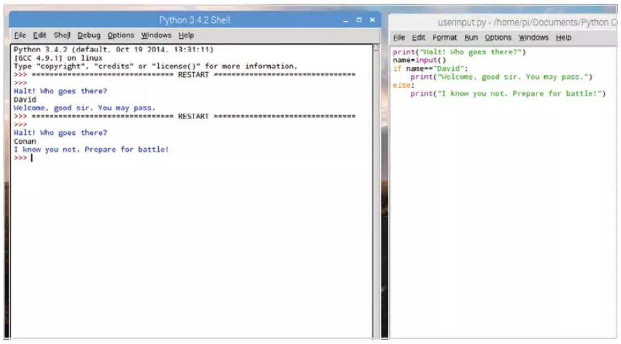
```
if name=="David":
  print("Welcome, good sir. You may pass.")
else:
  print("I know you not. Prepare for battle!")
```


userinput.py - /home/pi/Documents/Python Code/userinput.py (3.4.2)
File Edit Format Run Options Windows Help

```
print("Halt! Who goes there?")
name=input()
if name=="David":
    print("Welcome, good sir. You may pass.")
else:
    print("I know you not. Prepare for battle!")
```

STEP 8

What you've created here is a condition, which we will cover soon. In short, we're using the input from the user and measuring it against a condition. So, if the user enters David as their name, the guard will allow them to pass unhindered. Else, if they enter a name other than David, the guard challenges them to a fight.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
userinput.py - /home/pi/Documents/Python C
File Edit Format Run Options Windows Help

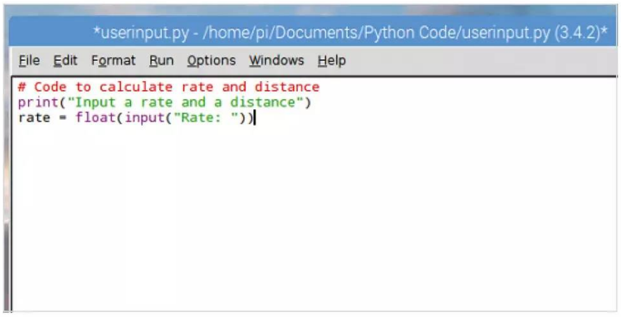
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Halt! Who goes there?
David
Welcome, good sir. You may pass.
>>> ================================ RESTART ================================
>>>
Halt! Who goes there?
Conan
I know you not. Prepare for battle!
>>>
```

```
print("Halt! Who goes there?")
name=input()
if name=="David":
    print("Welcome, good sir. You may pass.")
else:
    print("I know you not. Prepare for battle!")
```

STEP 9

Just as you learned previously, any input from a user is automatically a string, so you need to apply a TypeCast in order to turn it into something else. This creates some interesting additions to the input command. For example:

```
# Code to calculate rate and distance
print("Input a rate and a distance")
rate = float(input("Rate: "))
```


userinput.py - /home/pi/Documents/Python Code/userinput.py (3.4.2)
File Edit Format Run Options Windows Help

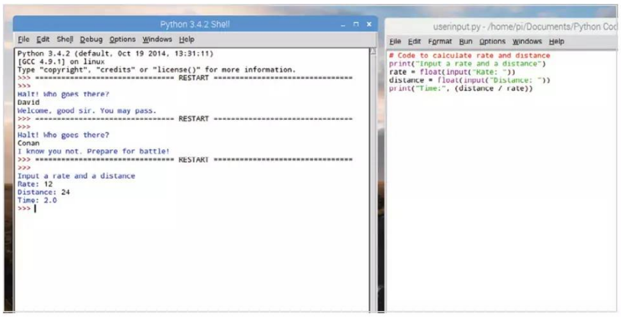
```
# Code to calculate rate and distance
print("Input a rate and a distance")
rate = float(input("Rate: "))
```

STEP 10

To finalise the rate and distance code, we can add:

```
distance = float(input("Distance: "))
print("Time:", (distance / rate))
```

Save and execute the code and enter some numbers. Using the float(input element, we've told Python that anything entered is a floating point number rather than a string.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
userinput.py - /home/pi/Documents/Python Cod
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Halt! Who goes there?
David
Welcome, good sir. You may pass.
>>> ================================ RESTART ================================
>>>
Halt! Who goes there?
Conan
I know you not. Prepare for battle!
>>> ================================ RESTART ================================
>>>
Input a rate and a distance
Rate: 12
Distance: 24
Time: 2.0
>>>
```

```
# Code to calculate rate and distance
print("Input a rate and a distance")
rate = float(input("Rate: "))
distance = float(input("Distance: "))
print("Time:", (distance / rate))
```

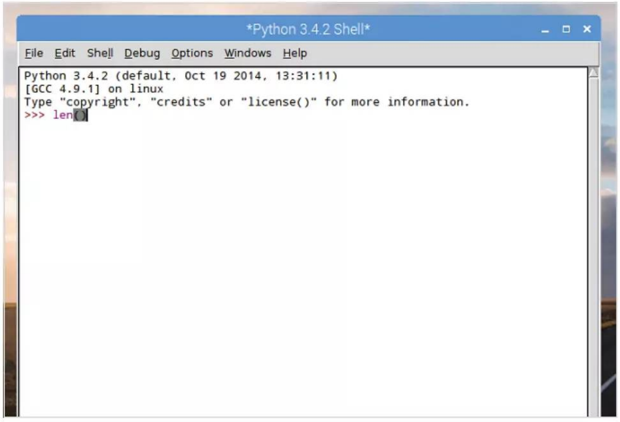
Creating Functions

Now that you've mastered the use of variables and user input, the next step is to tackle functions. You've already used a few functions, such as the print command but Python enables you to define your own functions.

FUNKY FUNCTIONS

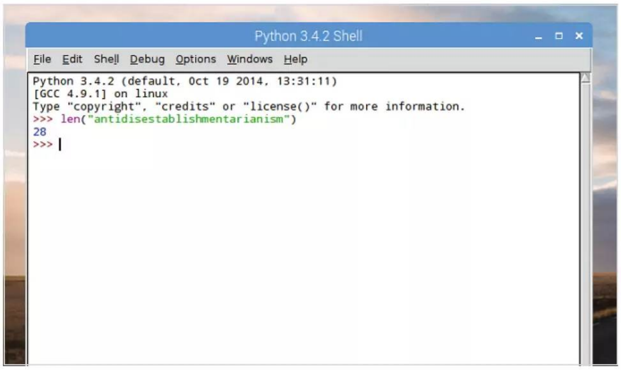
A function is a command that you enter into Python to do something. It's a little piece of self-contained code that takes data, works on it and then returns the result.

STEP 1 It's not just data that a function works on. They can do all manner of useful things in Python, such as sort data, change items from one format to another and check the length or type of items. Basically, a function is a short word that's followed by brackets. For example, **len()**, **list()** or **type()**.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> len()
```

STEP 2 A function takes data, usually a variable, works on it depending on what the function is programmed to do and returns the end value. The data being worked on goes inside the brackets, so if you wanted to know how many letters are in the word antidisestablishmentarianism, then you'd enter: `len("antidisestablishmentarianism")` and the number 28 would return.

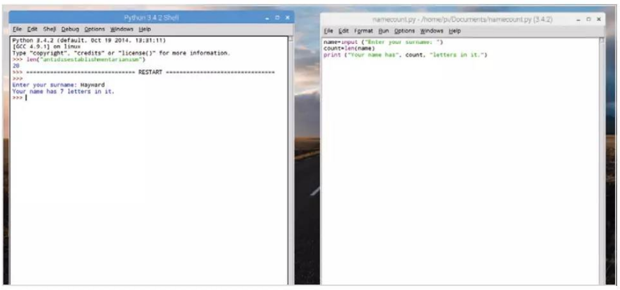

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>> |
```

STEP 3 You can pass variables through functions in much the same manner. Let's assume you want the number of letters in a person's surname, you could use the following code (enter the text editor for this example):

```
name=input ("Enter your surname: ")
count=len(name)
print ("Your surname has", count, "letters in
it.")
```

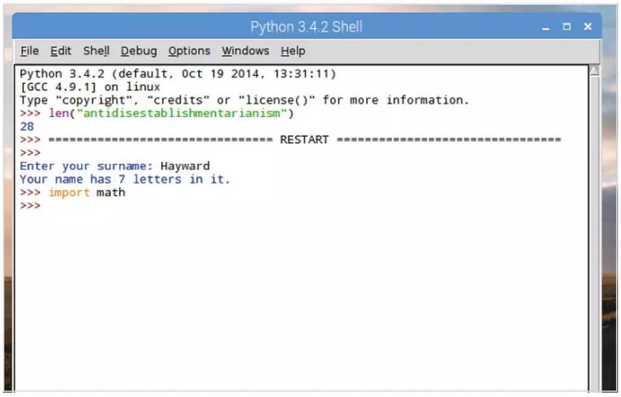
Press F5 and save the code to execute it.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
namecount.py - /home/pi/Documents/namecount.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>> ================================ RESTART ================================
>>>
Enter your surname: Hayward
Your name has 7 letters in it.
>>> |
```

```
name=input ("Enter your surname: ")
count=len(name)
print ("Your name has", count, "letters in it.")
```

STEP 4 Python has tens of functions built into it, far too many to get into in the limited space available here. However, to view the list of built-in functions available to Python 3, navigate to **www.docs.python.org/3/library/functions.html**. These are the predefined functions, but since users have created many more, they're not the only ones available.

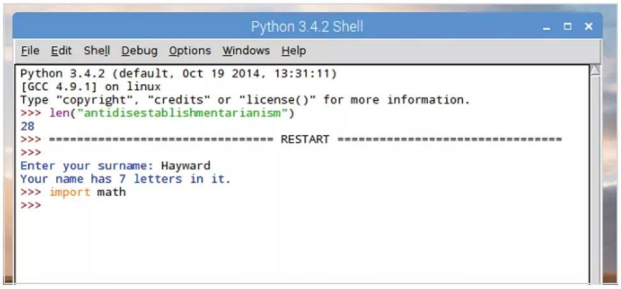

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>> ================================ RESTART ================================
>>>
Enter your surname: Hayward
Your name has 7 letters in it.
>>> import math
>>>
```

STEP 5 Additional functions can be added to Python through modules. Python has a vast range of modules available that can cover numerous programming duties. They add functions and can be imported as and when required. For example, to use advanced mathematics functions enter:

```
import math
```

Once entered, you have access to all the math module functions.

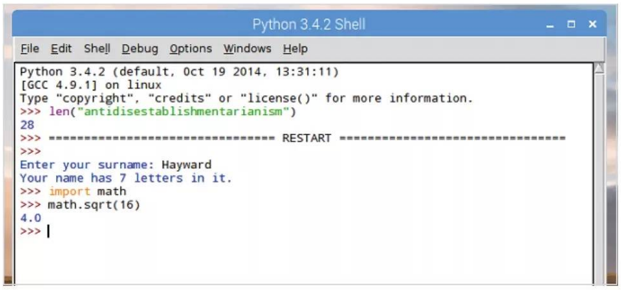

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>> ================================ RESTART ================================
>>>
Enter your surname: Hayward
Your name has 7 letters in it.
>>> import math
>>>
```

STEP 6 To use a function from a module enter the name of the module followed by a full stop, then the name of the function. For instance, using the math module, since you've just imported it into Python, you can utilise the square root function. To do so, enter:

```
math.sqrt(16)
```

You can see that the code is presented as module.function(data).


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> len("antidisestablishmentarianism")
28
>>> ================================ RESTART ================================
>>>
Enter your surname: Hayward
Your name has 7 letters in it.
>>> import math
>>> math.sqrt(16)
4.0
>>> |
```

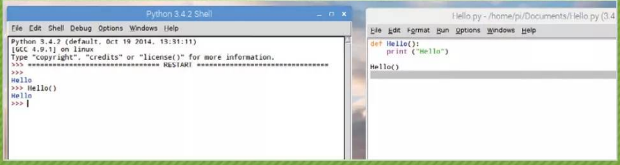
FORGING FUNCTIONS

There are many different functions you can import created by other Python programmers and you will undoubtedly come across some excellent examples in the future; you can also create your own with the def command.

STEP 1 Choose File > New File to enter the editor, let's create a function called Hello, that greets a user. Enter:

```
def Hello():
   print ("Hello")
Hello()
```

Press F5 to save and run the script. You can see Hello in the Shell, type in Hello() and it returns the new function.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Hello.py - /home/pi/Documents/Hello.py (3.4
File Edit Format Run Options Windows Help

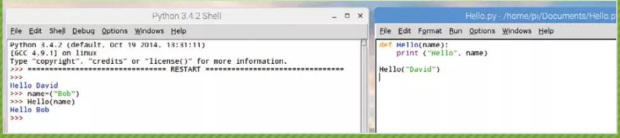
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello
>>> Hello()
Hello
>>> |
```

```
def Hello():
    print ("Hello")

Hello()
```

STEP 2 Let's now expand the function to accept a variable, the user's name for example. Edit your script to read:

```
def Hello(name):
   print ("Hello", name)
Hello("David")
```

This will now accept the variable name, otherwise it prints Hello David. In the Shell, enter: name=("Bob"), then: Hello(name). Your function can now pass variables through it.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Hello.py - /home/pi/Documents/Hello.p
File Edit Format Run Options Windows Help

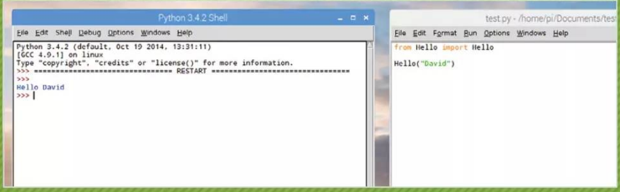
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello David
>>> name=("Bob")
>>> Hello(name)
Hello Bob
>>>
```

```
def Hello(name):
    print ("Hello", name)

Hello("David")
|
```

STEP 3 To modify it further, delete the Hello("David") line, the last line in the script and press Ctrl+S to save the new script. Close the Editor and create a new file (File > New File). Enter the following:

```
from Hello import Hello
Hello("David")
```

Press F5 to save and execute the code.

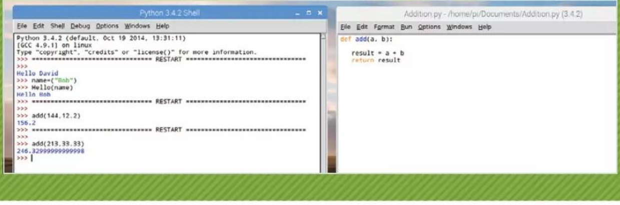

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
test.py - /home/pi/Documents/tes
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello David
>>> |
```

```
from Hello import Hello

Hello("David")
```

STEP 4 What you've just done is import the Hello function from the saved Hello.py program and then used it to say hello to David. This is how modules and functions work: you import the module then use the function. Try this one, and modify it for extra credit:

```
def add(a, b):
 result = a + b
 return result
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Addition.py - /home/pi/Documents/Addition.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Hello David
>>> name=("Bob")
>>> Hello(name)
Hello Bob
>>> ================================ RESTART ================================
>>>
>>> add(144,12.2)
156.2
>>> ================================ RESTART ================================
>>>
>>> add(213,33.33)
246.32999999999998
>>> |
```

```
def add(a, b):

   result = a + b
   return result
```

Conditions and Loops

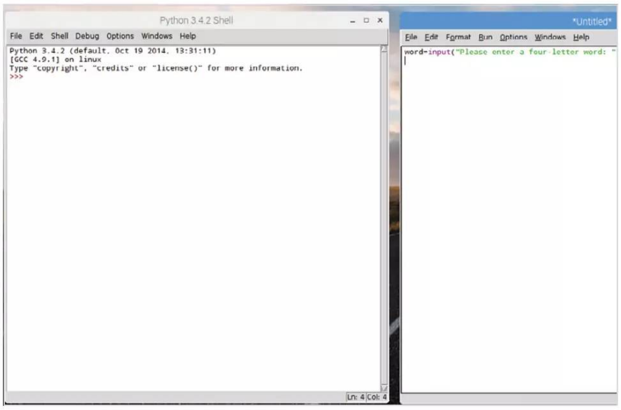
Conditions and loops are what makes a program interesting; they can be simple or rather complex. How you use them depends greatly on what the program is trying to achieve; they could be the number of lives left in a game or just displaying a countdown.

TRUE CONDITIONS

Keeping conditions simple to begin with makes learning to program a more enjoyable experience. Let's start then by checking if something is TRUE, then doing something else if it isn't.

STEP 1 Let's create a new Python program that will ask the user to input a word, then check it to see if it's a four-letter word or not. Start with File > New File, and begin with the input variable:

```
word=input(“Please enter a four-letter word: “)
```

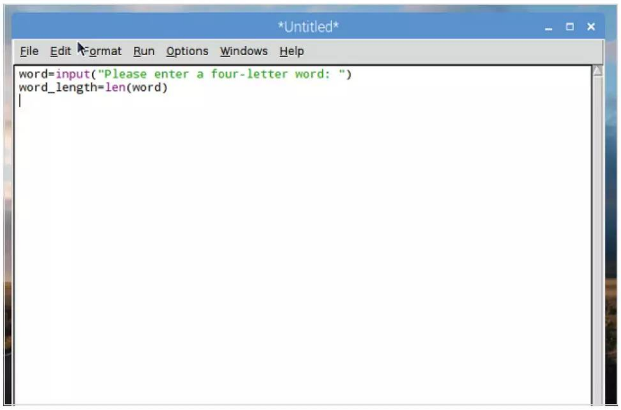

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Ln: 4 Col: 4
Untitled
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

```
word=input("Please enter a four-letter word: "
```

STEP 2 Now we can create a new variable, then use the len function and pass the word variable through it to get the total number of letters the user has just entered:

```
word=input(“Please enter a four-letter word: “)
word_length=len(word)
```

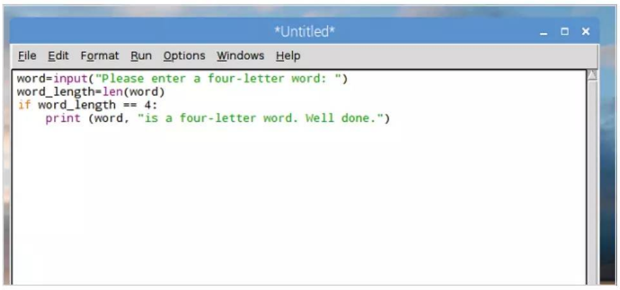

Untitled
File Edit Format Run Options Windows Help

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
```

STEP 3 Now you can use an if statement to check if the word_length variable is equal to four and print a friendly conformation if it applies to the rule:

```
word=input(“Please enter a four-letter word: “)
word_length=len(word)
if word_length == 4:
   print (word, “is a four-letter word. Well done.”)
```

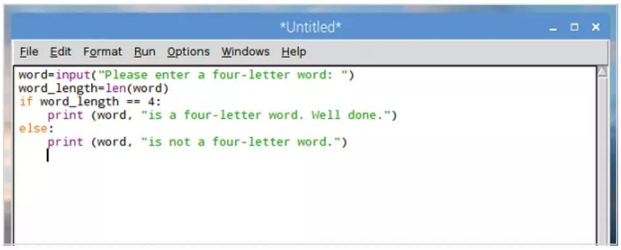
The double equal sign (==) means check if something is equal to something else.


Untitled
File Edit Format Run Options Windows Help

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
    print (word, "is a four-letter word. Well done.")
```

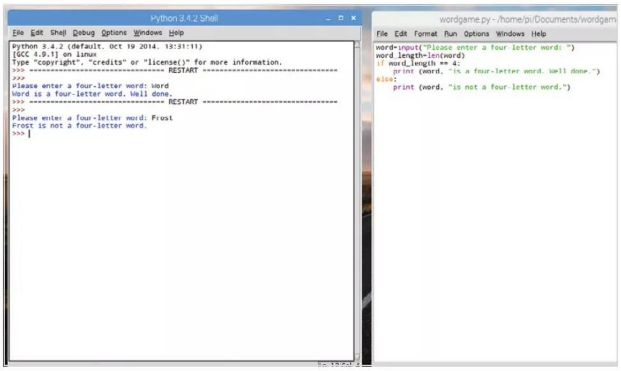
STEP 4 The colon at the end of IF tells Python that if this statement is true do everything after the colon that's indented. Next, move the cursor back to the beginning of the Editor:

```
word=input(“Please enter a four-letter word: “)
word_length=len(word)
if word_length == 4:
   print (word, “is a four-letter word. Well 
done.”)
else:
   print (word, “is not a four-letter word.”)
```


Untitled
File Edit Format Run Options Windows Help

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
    print (word, "is a four-letter word. Well done.")
else:
    print (word, "is not a four-letter word.")
```

STEP 5 Press F5 and save the code to execute it. Enter a four-letter word in the Shell to begin with, you should have the returned message that it's the word is four letters. Now press F5 again and rerun the program but this time enter a five-letter word. The Shell will display that it's not a four-letter word.

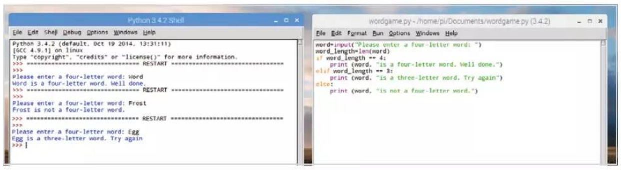

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
wordgame.py - /home/pi/Documents/wordgam
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Please enter a four-letter word: Word
Word is a four-letter word. Well done.
>>> ================================ RESTART ================================
>>>
Please enter a four-letter word: Frost
Frost is not a four-letter word.
>>> |
```

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
    print (word, "is a four-letter word. Well done.")
else:
    print (word, "is not a four-letter word.")
```

STEP 6 Now expand the code to include another conditions. Eventually, it could become quite complex. We've added a condition for three-letter words:

```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
   print (word, "is a four-letter word. Well done.")
elif word_length == 3:
   print (word, "is a three-letter word. Try again.")
else:
   print (word, "is not a four-letter word.")
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
wordgame.py - /home/pi/Documents/wordgame.py (3.4.2)
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
Please enter a four-letter word: Word
Word is a four-letter word. Well done.
>>> ================================ RESTART ================================
>>>
Please enter a four-letter word: Frost
Frost is not a four-letter word.
>>> ================================ RESTART ================================
>>>
Please enter a four-letter word: Egg
Egg is a three-letter word. Try again
>>> |
```

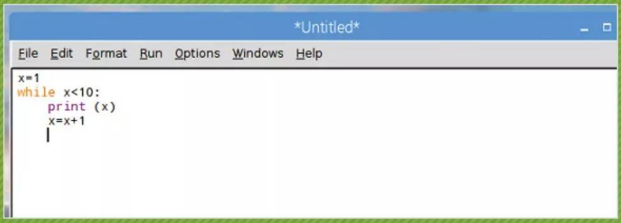
```
word=input("Please enter a four-letter word: ")
word_length=len(word)
if word_length == 4:
    print (word, "is a four-letter word. Well done.")
elif word_length == 3:
    print (word, "is a three-letter word. Try again")
else:
    print (word, "is not a four-letter word.")
```

LOOPS

A loop looks quite similar to a condition but they are somewhat different in their operation. A loop will run through the same block of code a number of times, usually with the support of a condition.

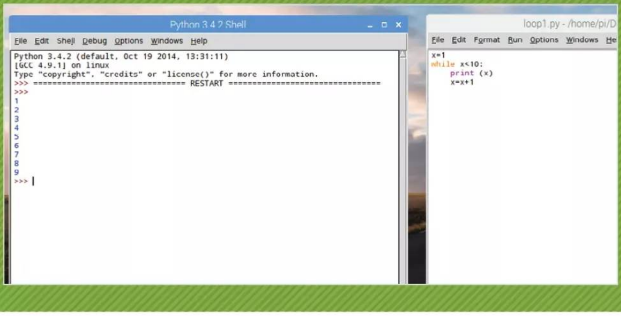
STEP 1 Let's start with a simple While statement. Like IF, this will check to see if something is TRUE, then run the indented code:

```
x = 1
while x < 10:
  print (x)
  x = x + 1
```


Untitled
File Edit Format Run Options Windows Help

```
x=1
while x<10:
    print (x)
    x=x+1
    |
```

STEP 2 The difference between if and while is when while gets to the end of the indented code, it goes back and checks the statement is still true. In our example x is less than 10. With each loop it prints the current value of x, then adds one to that value. When x does eventually equal 10 it stops.

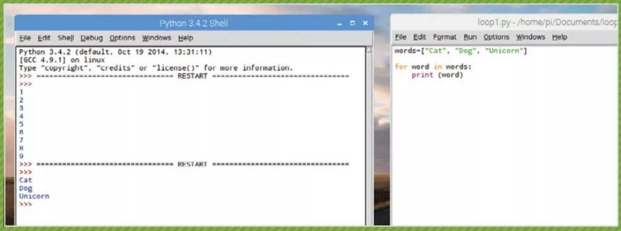

Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
loop1.py - /home/pi/D
File Edit Format Run Options Windows He

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
1
2
3
4
5
6
7
8
9
>>> |
```

```
x=1
while x<10:
    print (x)
    x=x+1
```

STEP 3 The For loop is another example. For is used to loop over a range of data, usually a list stored as variables inside square brackets. For example:

```
words=["Cat", "Dog", "Unicorn"]

for word in words:
   print (word)
```


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
loop1.py - /home/pi/Documents/loop
File Edit Format Run Options Windows Help

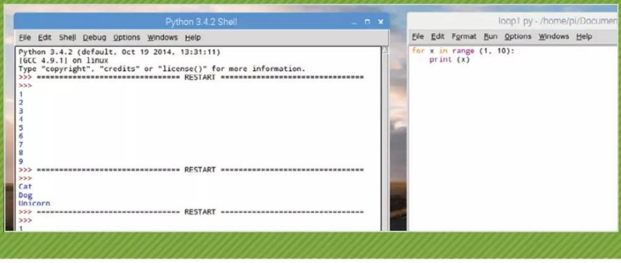
```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
1
2
3
4
5
6
7
8
9
>>> ================================ RESTART ================================
>>>
Cat
Dog
Unicorn
>>>
```

```
words=["Cat", "Dog", "Unicorn"]

for word in words:
    print (word)
```

STEP 4 The For loop can also be used in the countdown example by using the range function:

```
for x in range (1, 10):
 print (x)
```

The x=x+1 part isn't needed here because the range function creates a list between the first and last numbers used.


Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
loop1.py - /home/pi/Documen
File Edit Format Run Options Windows Help

```
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
1
2
3
4
5
6
7
8
9
>>> ================================ RESTART ================================
>>>
Cat
Dog
Unicorn
>>> ================================ RESTART ================================
>>>
1
```

```
for x in range (1, 10):
    print (x)
```

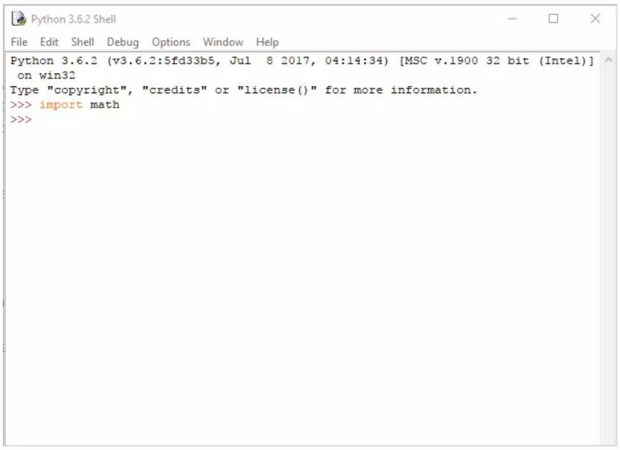
Python Modules

We've mentioned modules previously, (the Math module) but as modules are such a large part of getting the most from Python, it's worth dedicating a little more time to them. In this instance we're using the Windows version of Python 3.

MASTERING MODULES

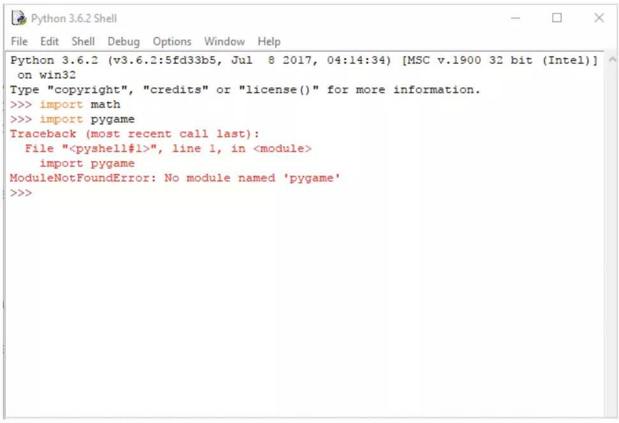
Think of modules as an extension that's imported into your Python code to enhance and extend its capabilities. There are countless modules available and as we've seen, you can even make your own.

STEP 1 Although good, the built-in functions within Python are limited. The use of modules, however, allows us to make more sophisticated programs. As you are aware, modules are Python scripts that are imported, such as import math.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> import math
>>>
```

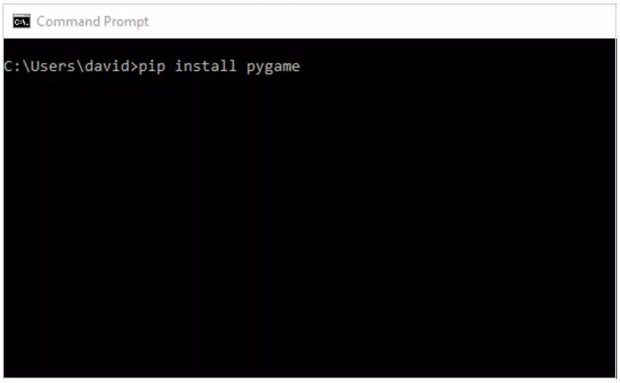
STEP 2 Some modules, especially on the Raspberry Pi, are included by default, the math module being a prime example. Sadly, other modules aren't always available. A good example on non-Pi platforms is the pygame module, which contains many functions to help create games. Try: **import pygame**.



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> import math
>>> import pygame
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    import pygame
ModuleNotFoundError: No module named 'pygame'
>>>
```

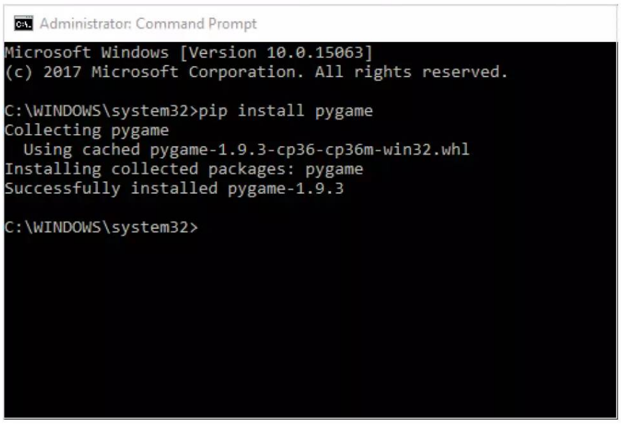
STEP 3 The result is an error in the IDLE Shell, as the pygame module isn't recognised or installed in Python. To install a module we can use PIP (Pip Installs Packages). Close down the IDLE Shell and drop into a command prompt or Terminal session. At an elevated admin command prompt, enter:

```
pip install pygame
```



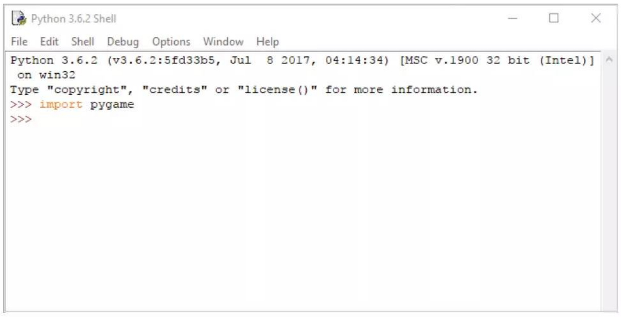
```
C:\Users\david>pip install pygame
```

STEP 4 The PIP installation requires an elevated status due it installing components at different locations. Windows users can search for CMD via the Start button and right-click the result then click Run as Administrator. Linux and Mac users can use the Sudo command, with sudo pip install package.



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>pip install pygame
Collecting pygame
  Using cached pygame-1.9.3-cp36-cp36m-win32.whl
Installing collected packages: pygame
Successfully installed pygame-1.9.3

C:\WINDOWS\system32>
```

STEP 5 Close the command prompt or Terminal and relaunch the IDLE Shell. When you now enter: `import pygame`, the module will be imported into the code without any problems. You'll find that most code downloaded or copied from the Internet will contain a module, mainstream of unique, these are usually the source of errors in execution due to them being missing.


Python 3.6.2 Shell
File Edit Shell Debug Options Window Help

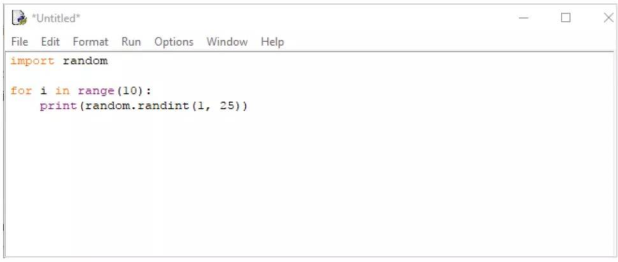
```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.1900 32 bit (Intel)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>> import pygame
>>>
```

STEP 6 The modules contain the extra code needed to achieve a certain result within your own code, as we've previously experimented with. For example:

```
import random
```

Brings in the code from the random number generator module. You can then use this module to create something like:

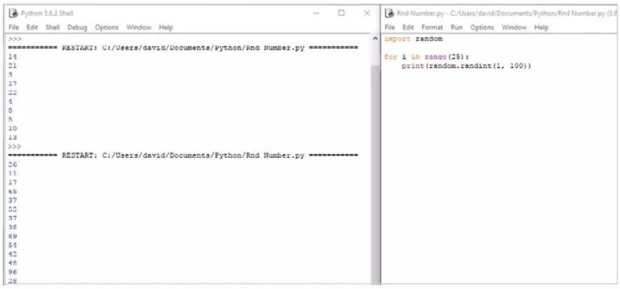
```
for i in range(10):
  print(random.randint(1, 25))
```


Untitled
File Edit Format Run Options Window Help

```
import random

for i in range(10):
    print(random.randint(1, 25))
```

STEP 7 This code, when saved and executed, will display ten random numbers from 1 to 25. You can play around with the code to display more or less, and from a great or lesser range. For example:

```
import random

for i in range(25):
  print(random.randint(1, 100))
```

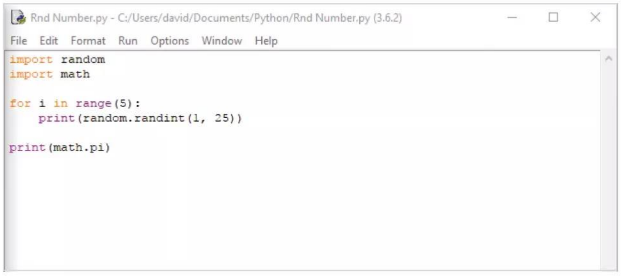

Python 3.6.2 Shell
File Edit Shell Debug Options Window Help
Rnd Number.py - C:/Users/david/Documents/Python/Rnd Number.py (3.6
File Edit Format Run Options Window Help

```
>>>
=========== RESTART: C:/Users/david/Documents/Python/Rnd Number.py ===========
14
21
3
17
22
4
8
5
10
13
>>>
=========== RESTART: C:/Users/david/Documents/Python/Rnd Number.py ===========
26
11
17
65
37
52
37
38
89
54
42
48
96
28
```

```
import random

for i in range(25):
    print(random.randint(1, 100))
```

STEP 8 Multiple modules can be imported within your code. To extend our example, use:

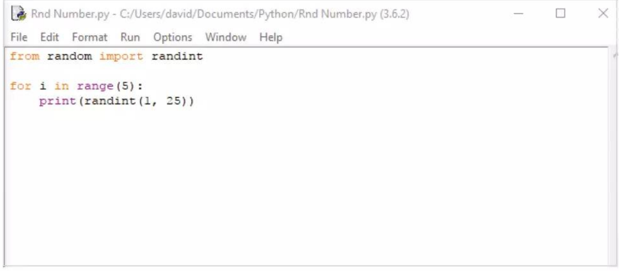
```
import random
import math

for I in range(5):
  print(random.randint(1, 25))

print(math.pi)
```


Rnd Number.py - C:/Users/david/Documents/Python/Rnd Number.py (3.6.2)
File Edit Format Run Options Window Help

```
import random
import math

for i in range(5):
    print(random.randint(1, 25))

print(math.pi)
```

STEP 9 The result is a string of random numbers followed by the value of Pi as pulled from the math module using the print(math.pi) function. You can also pull in certain functions from a module by using the from and import commands, such as:

```
from random import randint

for i in range(5):
  print(randint(1, 25))
```

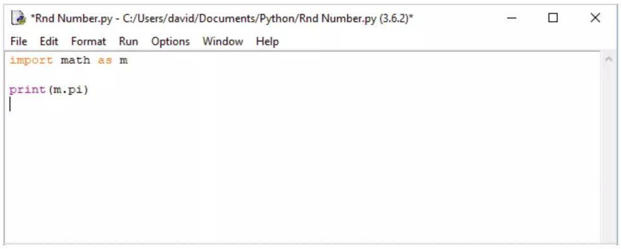

Rnd Number.py - C:/Users/david/Documents/Python/Rnd Number.py (3.6.2)
File Edit Format Run Options Window Help

```
from random import randint

for i in range(5):
    print(randint(1, 25))
```

STEP 10 This helps create a more streamlined approach to programming. You can also use import module*, which will import everything defined within the named module. However, it's often regarded as a waste of resources but it works nonetheless. Finally, modules can be imported as aliases:

```
import math as m

print(m.pi)
```

Of course, adding comments helps to tell others what's going on.


Rnd Number.py - C:/Users/david/Documents/Python/Rnd Number.py (3.6.2)
File Edit Format Run Options Window Help

```
import math as m

print(m.pi)
```

Say Hello to C++

C++ is a high level programming language that's used in a multitude of technologies. Everything from your favourite mobile app, console and PC game to entire operating systems, all are developed using C++ and a collection of software development kits and custom libraries.

C++ is the driving force behind most of what you use on a daily basis, which makes it a complex and extraordinarily powerful language to get to grips with. In this section, we look at how to install a C++ IDE and compiler on your computer.

Why C++?

C++ is one of the most popular programming languages available today. Originally called C with Classes, the language was renamed C++ in 1983. It's an extension of the original C language and is a general purpose object-oriented (OOP) environment.

C EVERYTHING

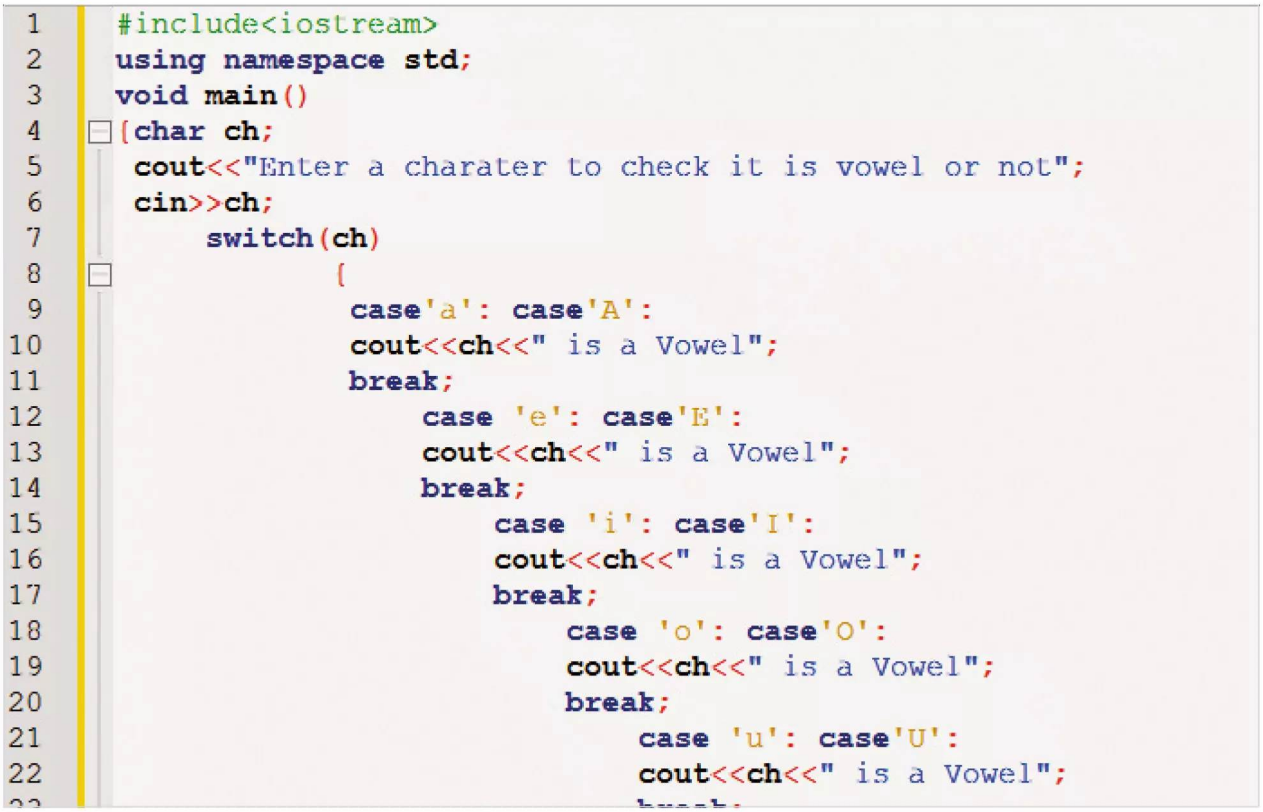
Due to how complex the language can be, and its power and performance, C++ is often used to develop games, programs, device drivers and even entire operating systems.

Dating back to 1979, the start of the golden era of home computing, C++, or rather C with Classes, was the brainchild of Danish computer scientist Bjarne Stroustrup while working on his PhD thesis. Stroustrup's plan was to further the original C language, which was widely used since the early seventies.

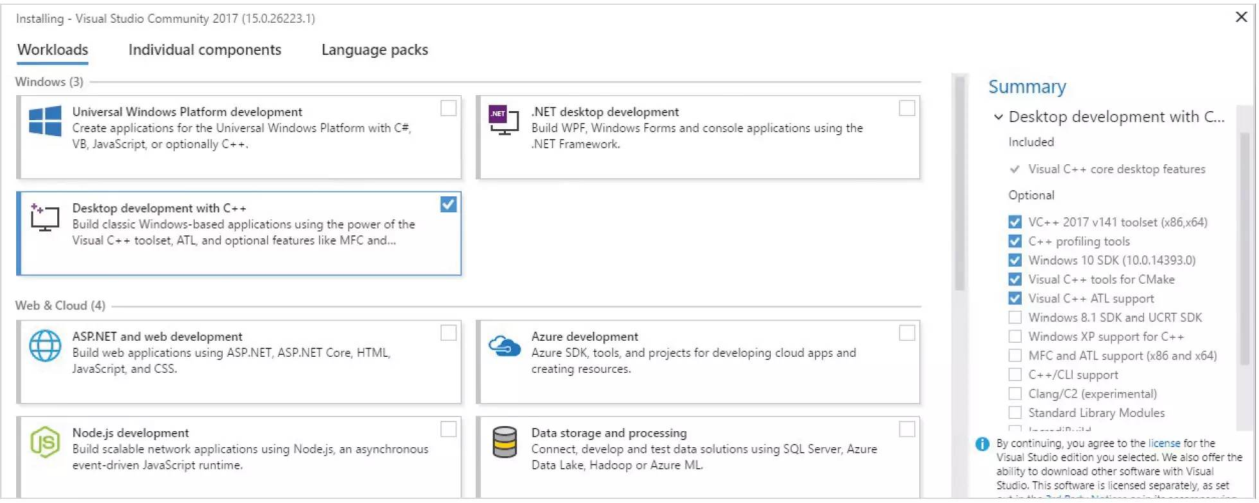
C++ proved to be popular among the developers of the '80s, since it was a much easier environment to get to grips with and more importantly, it was 99% compatible with the original C language. This meant that it could be used beyond the mainstream computing labs and by regular people who didn't have access to the mainframes and large computing data centres.

C++'s impact in the digital world is immense. Many of the programs, applications, games and even operating systems are coded using C++. For example, all of Adobe's major applications, such as Photoshop, InDesign and so on, are developed in C++. You will find that the browser you surf the Internet with is written in C++, as well as Windows 10, Microsoft Office and the backbone to Google's search engine. Apple's macOS is written largely in C++ (with some

C++ code is much faster than that of Python.



```
#include<iostream>
using namespace std;
void main()
{char ch;
 cout<<"Enter a charater to check it is vowel or not";
 cin>>ch;
     switch(ch)
            {
             case'a': case'A':
             cout<<ch<<" is a Vowel";
             break;
                 case 'e': case'E':
                 cout<<ch<<" is a Vowel";
                 break;
                     case 'i': case'I':
                     cout<<ch<<" is a Vowel";
                     break;
                         case 'o': case'O':
                         cout<<ch<<" is a Vowel";
                         break;
                             case 'u': case'U':
                             cout<<ch<<" is a Vowel";
```



Installing - Visual Studio Community 2017 (15.0.26223.1)

Workloads | Individual components | Language packs

Windows (3)

Universal Windows Platform development ☐
Create applications for the Universal Windows Platform with C#, VB, JavaScript, or optionally C++.

.NET desktop development ☐
Build WPF, Windows Forms and console applications using the .NET Framework.

Desktop development with C++ ☑
Build classic Windows-based applications using the power of the Visual C++ toolset, ATL, and optional features like MFC and...

Web & Cloud (4)

ASP.NET and web development ☐
Build web applications using ASP.NET, ASP.NET Core, HTML, JavaScript, and CSS.

Azure development ☐
Azure SDK, tools, and projects for developing cloud apps and creating resources.

Node.js development ☐
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.

Data storage and processing ☐
Connect, develop and test data solutions using SQL Server, Azure Data Lake, Hadoop or Azure ML.

Summary

Desktop development with C...

Included

✓ Visual C++ core desktop features

Optional

- ☑ VC++ 2017 v141 toolset (x86,x64)
- ☑ C++ profiling tools
- ☑ Windows 10 SDK (10.0.14393.0)
- ☑ Visual C++ tools for CMake
- ☑ Visual C++ ATL support
- ☐ Windows 8.1 SDK and UCRT SDK
- ☐ Windows XP support for C++
- ☐ MFC and ATL support (x86 and x64)
- ☐ C++/CLI support
- ☐ Clang/C2 (experimental)
- ☐ Standard Library Modules

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set

Microsoft's Visual Studio is a great, free environment to learn C++ in.

other languages mixed in depending on the function) and the likes of NASA, SpaceX and even CERN use C++ for various applications, programs, controls and umpteen other computing tasks.

C++ is also extremely efficient and performs well across the board as well as being an easier addition to the core C language. This higher level of performance over other languages, such as Python, BASIC and such, makes it an ideal development environment for modern computing, hence the aforementioned companies using it so widely.

While Python is a great programming language to learn, C++ puts the developer in a much wider world of coding. By mastering C++, you can find yourself developing code for the likes of Microsoft, Apple and so on. Generally, C++ developers enjoy a higher salary than programmers of some other languages and due to its versatility, the C++ programmer can move between jobs and companies without the need to relearn anything specific. However, Python is an easier language to begin with. If you're completely new to programming then we would recommend you begin with Python and spend some time getting to grips with programming structure and the many ways and means in which you find a solution to a problem through programming. Once you can happily power up your computer and whip out a Python program with one hand tied behind your back, then move on to C++. Of course, there's nothing stopping you from jumping straight into C++; if you feel up to the task, go for it.

Getting to use C++ is as easy as Python, all you need is the right set of tools in which to communicate with the computer in C++ and you can start your journey. A C++ IDE is free of charge, even the immensely powerful Visual Studio from Microsoft is freely available to download and use. You can get into C++ from any operating system, be it macOS, Linux, Windows or even mobile platforms.

Just like Python, to answer the question of Why C++ is the answer is because it's fast, efficient and developed by most of the applications you regularly use. It's cutting edge and a fantastic language to master.

Microsoft
Windows
Mac
debian
ubuntu

Indeed, the operating system you're using is written in C++.

Equipment Needed

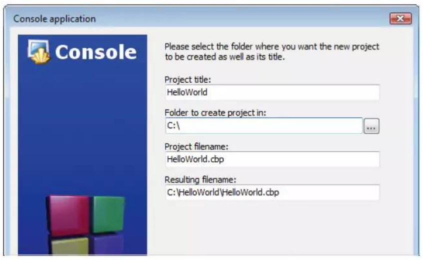
You don't need to invest a huge amount of money in order to learn C++ and you don't need an entire computing lab at your disposal either. Providing you have a fairly modern computer, everything else is freely available.

C++ SETUPS

Most, if not all, operating systems have C++ in their code, so it stands to reason that you can learn to program in C++ no matter what OS you're currently using.

☐ COMPUTER

Unless you fancy writing out your C++ code by hand on a sheet of paper (which is something many older coders used to do), a computer is an absolute must have component. PC users can have any recent Linux distro or Windows OS, Mac users the latest macOS.



Console application

Console

Please select the folder where you want the new project to be created as well as its title.

Project title: HelloWorld

Folder to create project in: C:\

Project filename: HelloWorld.cbp

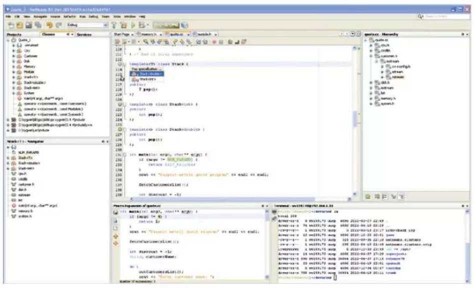
Resulting filename: C:\HelloWorld\HelloWorld.cbp

☐ AN IDE

Just as with Python, an IDE is used to enter and execute your C++ code. Many IDEs come with extensions and plugins that help make it work better, or add an extra level of functionality. Often, an IDE provides enhancements depending on the core OS being used, such as being enhanced for Windows 10.

☐ COMPILER

A compiler is a program that converts the C++ language into binary, so that the computer can understand. While some IDEs come with a compiler built in, others don't. Code::Blocks is our favourite IDE that comes with a C++ compiler as part of the package. More on this later.

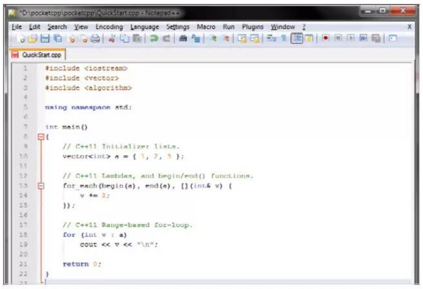


☐ TEXT EDITOR

Some programmers much prefer to use a text editor to assemble their C++ code before running it through a compiler. Essentially you can use any text editor to write code, just save it with a .cpp extension. However, Notepad++ is one of the best code text editors available.

☐ INTERNET ACCESS

While it's entirely possible to learn how to code on a computer that's not attached to the Internet, it's extraordinarily difficult. You need to install relevant software, keep it up to date, install any extras or extensions and look for help when coding. All of these require access to the Internet.



```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main()
{
    // C++11 Initializer lists.
    vector<int> a = { 1, 2, 3 };

    // C++11 Lambdas, and begin/end() functions.
    for_each(begin(a), end(a), [](int& v) {
        v *= 2;
    });

    // C++11 Range-based for-loop.
    for (int v : a)
        cout << v << "\n";

    return 0;
}
```

☐ TIME AND PATIENCE

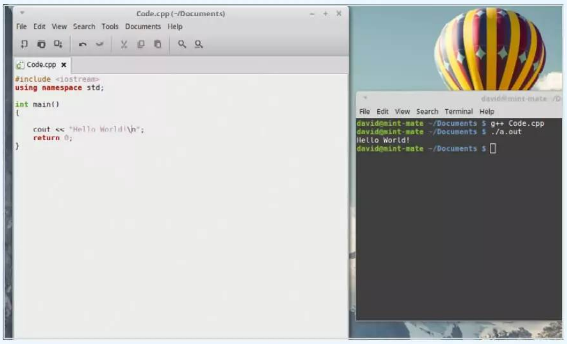
Yes, as with Python, you're going to need to set aside significant time to spend on learning how to code in C++. Sadly, unless you're a genius, it's not going to happen overnight, or even a week. A good C++ coder has spent many years honing their craft, so be patient, start small and keep learning.

OS SPECIFIC NEEDS

C++ will work in any operating system but getting all the necessary pieces together can be confusing to a newcomer. Here are some OS specifics for C++.

LINUX

Linux users are lucky in that they already have a compiler and text editor built into their operating system. Any text editor allows you to type out your C++ code, when it's saved with a .cpp extension, use g++ to compile it.

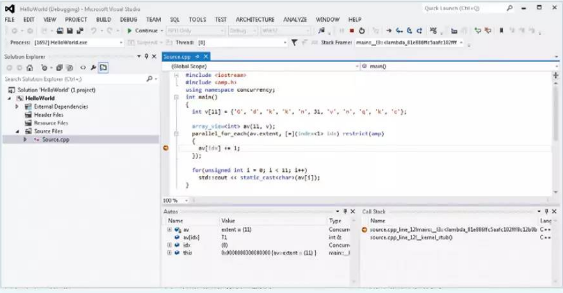


```cpp
#include <iostream>
using namespace std;

int main()
{

    cout << "Hello World!\n";
    return 0;
}
```

```
david@mint-mate ~/Documents $ g++ Code.cpp
david@mint-mate ~/Documents $ ./a.out
Hello World!
david@mint-mate ~/Documents $
```

WINDOWS

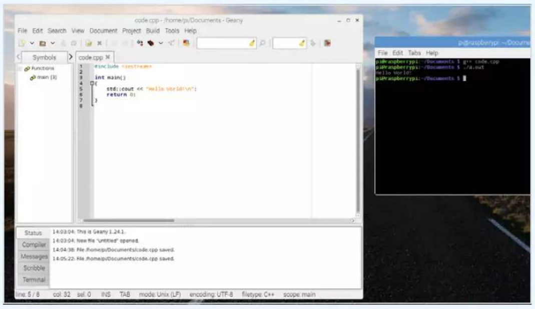
We have mentioned previously that a good IDE is Microsoft's Visual Studio. However, a better IDE and compiler is Code::Blocks, which is regularly kept up to date with a new release twice a year. Otherwise Windows users can enter their code in Notepad++, then compile it with MinGW as used by Code::Blocks.



```cpp
#include <iostream>
#include <amp.h>
using namespace concurrency;
int main()
{
  int v[11] = {'G', 'd', 'k', 'k', 'n', 31, 'v', 'n', 'q', 'k', 'c'};

  array_view<int> av(11, v);
  parallel_for_each(av.extent, [=](index<1> idx) restrict(amp)
  {
    av[idx] += 1;
  });

  for(unsigned int i = 0; i < 11; i++)
    std::cout << static_cast<char>(av[i]);
}
```

RASPBERRY PI

The Raspberry Pi's operating system is Raspbian, which is Linux based. Therefore, you're able to write your code out using a text editor, then compile it with g++ as you would in any other Linux distro.



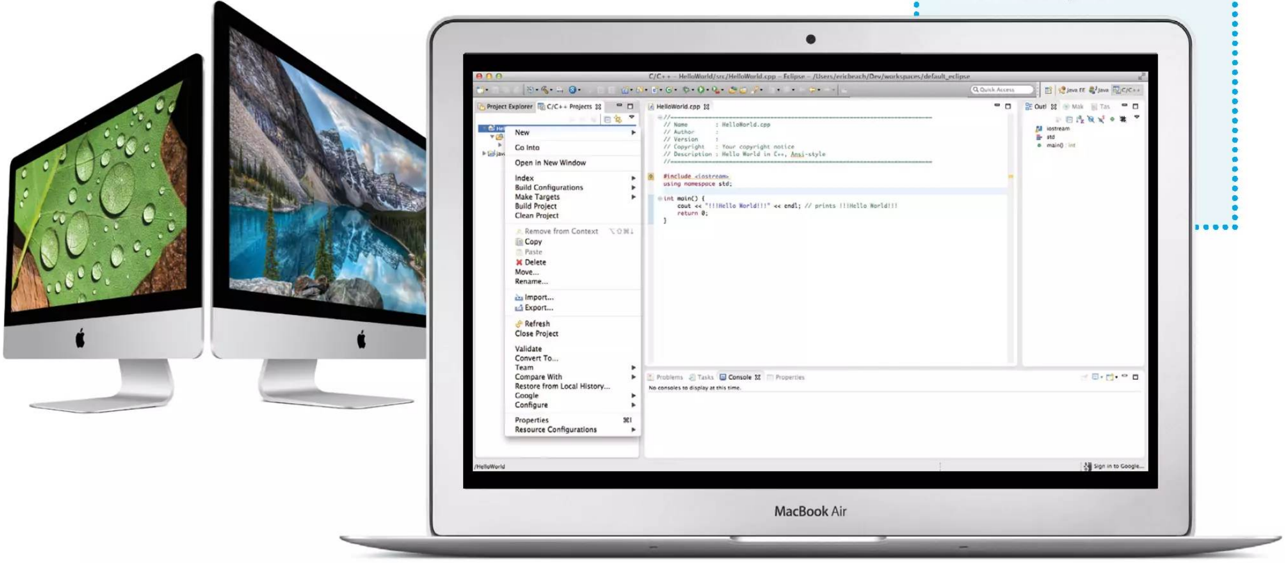
```cpp
#include <iostream>

int main()
{
    std::cout << "Hello World!\n";
    return 0;
}
```

```
pi@raspberrypi:~/Documents $ g++ code.cpp
pi@raspberrypi:~/Documents $ ./a.out
Hello World!
pi@raspberrypi:~/Documents $
```

```
14:03:04: This is Geany 1.24.1.
14:03:04: New file "untitled" opened.
14:04:38: File /home/pi/Documents/code.cpp saved.
14:05:22: File /home/pi/Documents/code.cpp saved.
```

MAC

Mac owners will need to download and install Xcode to be able to compile their C++ code natively. Other options for the macOS include Netbeans, Eclipse or Code::Blocks. Note: the latest Code::Blocks isn't available for Mac due to a lack of Mac developers.



```cpp
//============================================================================
// Name        : HelloWorld.cpp
// Author      :
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================================

#include <iostream>
using namespace std;

int main() {
	cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!
	return 0;
}
```

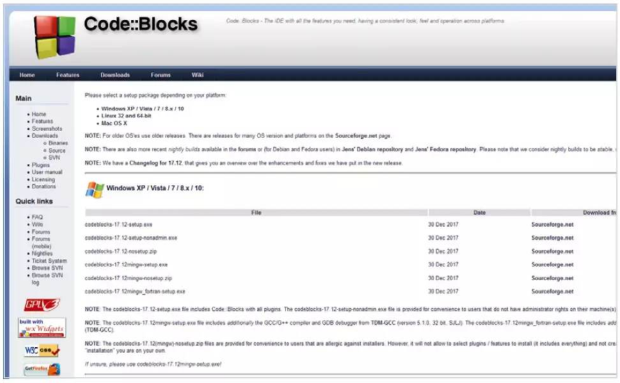
How to Set Up C++ in Windows

Windows users have a wealth of choice when it comes to programming in C++. There are plenty of IDEs and compilers available, including Visual Studio from Microsoft. However, in our opinion, the best C++ IDE to begin with is Code::Blocks.

CODE::BLOCKS

Code::Blocks is a free C++, C and Fortran IDE that's feature rich and easily extendible with plug-ins. It's easy to use, comes with a compiler and has a vibrant community behind it.

STEP 1 Start by visiting the Code::Blocks download site, at **www.codeblocks.org/downloads**. From there, click on the 'Download the binary releases' link to be taken to the latest downloadable version for Windows.



Code::Blocks

Code::Blocks - The IDE with all the features you need, having a consistent look, feel and operation across platforms

Please select a setup package depending on your platform:

- Windows XP / Vista / 7 / 8.x / 10
- Linux 32 and 64-bit
- Mac OS X

NOTE: For older OS'es use older releases. There are releases for many OS version and platforms on the **Sourceforge.net** page.

NOTE: There are also more recent *nightly* builds available in the **forums** or (for Debian and Fedora users) in **Jens' Debian repository** and **Jens' Fedora repository**. Please note that we consider nightly builds to be stable,

NOTE: We have a **Changelog for 17.12**, that gives you an overview over the enhancements and fixes we have put in the new release.

Windows XP / Vista / 7 / 8.x / 10:

File	Date	Download fr
codeblocks-17.12-setup.exe	30 Dec 2017	Sourceforge.net
codeblocks-17.12-setup-nonadmin.exe	30 Dec 2017	Sourceforge.net
codeblocks-17.12-nosetup.zip	30 Dec 2017	Sourceforge.net
codeblocks-17.12mingw-setup.exe	30 Dec 2017	Sourceforge.net
codeblocks-17.12mingw-nosetup.zip	30 Dec 2017	Sourceforge.net
codeblocks-17.12mingw_fortran-setup.exe	30 Dec 2017	Sourceforge.net

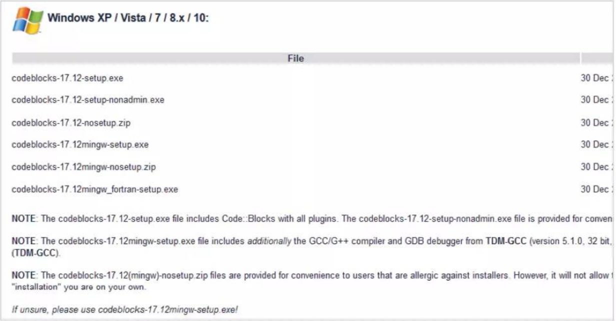
NOTE: The codeblocks-17.12-setup.exe file includes Code::Blocks with all plugins. The codeblocks-17.12-setup-nonadmin.exe file is provided for convenience to users that do not have administrator rights on their machine(s)

NOTE: The codeblocks-17.12mingw-setup.exe file includes additionally the GCC/G++ compiler and GDB debugger from TDM-GCC (version 5.1.0, 32 bit, SJLJ). The codeblocks-17.12mingw_fortran-setup.exe file includes add (TDM-GCC).

NOTE: The codeblocks-17.12(mingw)-nosetup.zip files are provided for convenience to users that are allergic against installers. However, it will not allow to select plugins / features to install (it includes everything) and not cre "installation" you are on your own.

If unsure, please use codeblocks-17.12mingw-setup.exe!

STEP 2 You can see that there are several Windows versions available. The one you want to download has 'mingw-setup.exe' at the end of the current version number. At the time of writing this is: codeblocks-17.12mingw-setup.exe. The difference is that the mingw-setup version includes a C++ compiler and debugger from TDM-GCC (a compiler suite).



Windows XP / Vista / 7 / 8.x / 10:

File	
codeblocks-17.12-setup.exe	30 Dec
codeblocks-17.12-setup-nonadmin.exe	30 Dec
codeblocks-17.12-nosetup.zip	30 Dec
codeblocks-17.12mingw-setup.exe	30 Dec
codeblocks-17.12mingw-nosetup.zip	30 Dec
codeblocks-17.12mingw_fortran-setup.exe	30 Dec

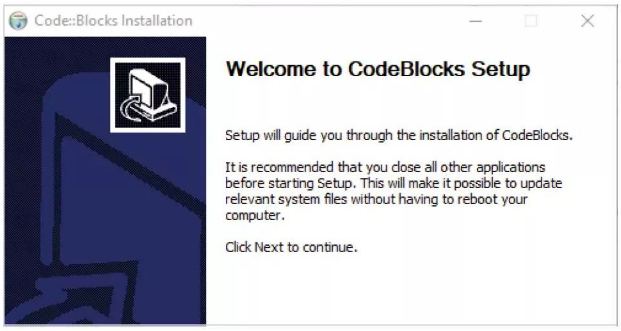
NOTE: The codeblocks-17.12-setup.exe file includes Code::Blocks with all plugins. The codeblocks-17.12-setup-nonadmin.exe file is provided for conven

NOTE: The codeblocks-17.12mingw-setup.exe file includes *additionally* the GCC/G++ compiler and GDB debugger from **TDM-GCC** (version 5.1.0, 32 bit, (TDM-GCC).

NOTE: The codeblocks-17.12(mingw)-nosetup.zip files are provided for convenience to users that are allergic against installers. However, it will not allow "installation" you are on your own.

If unsure, please use codeblocks-17.12mingw-setup.exe!

STEP 3 When you've located the file, click on the Sourceforge.net link at the end of the line and a download notification window appears; click on Save File to start the download and save the executable to your PC. Locate the downloaded Code::Blocks installer and double-click to start. Follow the on-screen instructions to begin the installation.



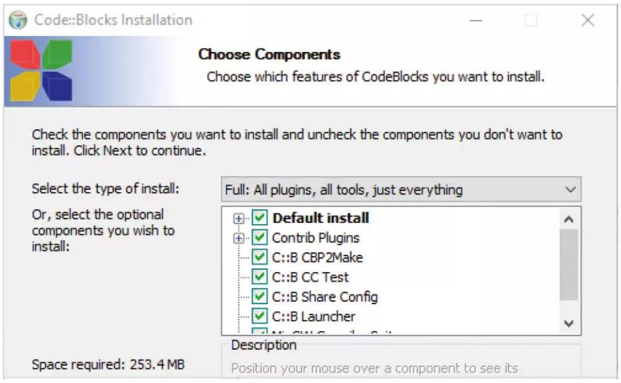
Welcome to CodeBlocks Setup

Setup will guide you through the installation of CodeBlocks.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

STEP 4 Once you've agreed to the licencing terms, there is a choice of installation options available. You can opt for a smaller install, missing out on some of the components but we would recommend you opt for the Full option as default.



Choose Components

Choose which features of CodeBlocks you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select the type of install: Full: All plugins, all tools, just everything

Or, select the optional components you wish to install:

- ☑ **Default install**
- ☑ Contrib Plugins
  - ☑ C::B CBP2Make
  - ☑ C::B CC Test
  - ☑ C::B Share Config
  - ☑ C::B Launcher

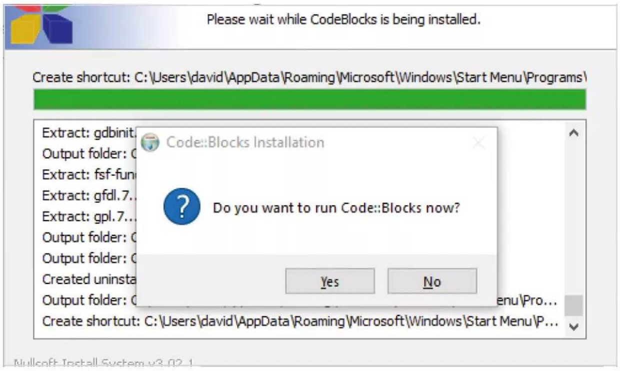
Space required: 253.4 MB

Description

Position your mouse over a component to see its

STEP 5

Next choose an install location for the Code::Blocks files. It's your choice, but the default will generally suffice, unless of course you have any special requirements. When you click Next, the install begins; when it's finished a notification pops up asking you if you want to start Code::Blocks now, so click Yes.



Please wait while CodeBlocks is being installed.

```
Create shortcut: C:\Users\david\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\
Extract: gdbinit.
Output folder: C
Extract: fsf-fun
Extract: gfdl.7..
Extract: gpl.7...
Output folder: C
Output folder: C
Created uninsta
Output folder: C                                                        enu\Pro...
Create shortcut: C:\Users\david\AppData\Roaming\Microsoft\Windows\Start Menu\P...
```

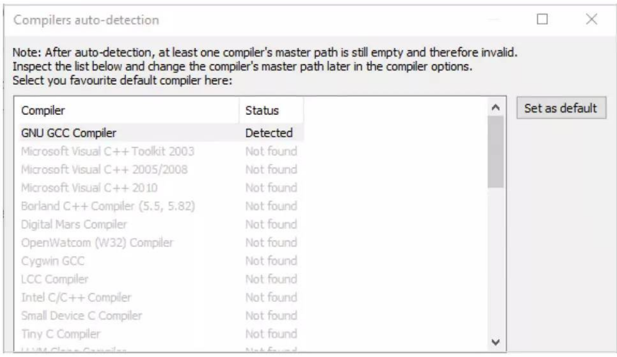
Code::Blocks Installation

Do you want to run Code::Blocks now?

Yes
No
Nullsoft Install System v3.02.1

STEP 6

The first time Code::Blocks loads it runs an auto-detect for any C++ compilers you may already have installed on you system. If you don't have any, click on the first detected option, GNU GCC Compiler, and click the Default button to set it as the system's C++ compiler. Click OK when you're ready to continue.



Compilers auto-detection

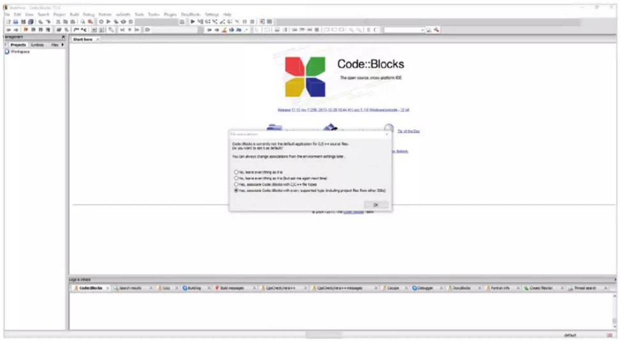
Note: After auto-detection, at least one compiler's master path is still empty and therefore invalid.
Inspect the list below and change the compiler's master path later in the compiler options.
Select you favourite default compiler here:

Compiler	Status
GNU GCC Compiler	Detected
Microsoft Visual C++ Toolkit 2003	Not found
Microsoft Visual C++ 2005/2008	Not found
Microsoft Visual C++ 2010	Not found
Borland C++ Compiler (5.5, 5.82)	Not found
Digital Mars Compiler	Not found
OpenWatcom (W32) Compiler	Not found
Cygwin GCC	Not found
LCC Compiler	Not found
Intel C/C++ Compiler	Not found
Small Device C Compiler	Not found
Tiny C Compiler	Not found

Set as default

STEP 7

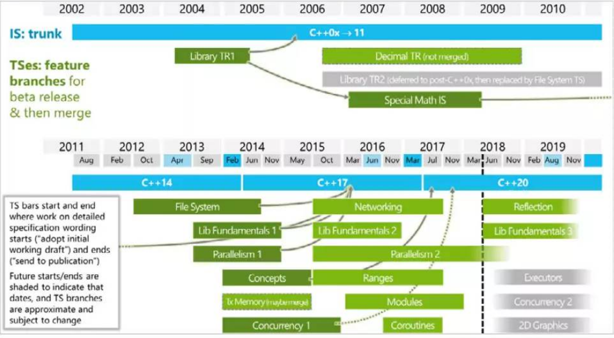
When the program starts another message appears, informing you that Code::Blocks is currently not the default application for C++ files. You a couple of options: to leave everything as it is or allow Code::Blocks to associate all C++ file types. Again, we would recommend you opt for the last choice to associate Code::Blocks with every supported file type.



Code::Blocks

STEP 8

Before you start using Code::Blocks it's worth explaining exactly why you need the added compiler. First, a compiler is a separate program that reads through your C++ code and checks it against the latest acceptable programming standards; this is why you need the most recently available compiler. This is currently C++17, with C++20 underway.

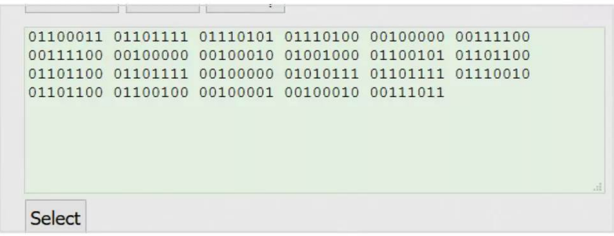


2002
2003
2004
2005
2006
2007
2008
2009
2010
IS: trunk
C++0x → 11
TSes: feature branches for beta release & then merge
Library TR1
Decimal TR (not merged)
Library TR2 (deferred to post-C++0x, then replaced by File System TS)
Special Math IS
2011
2012
2013
2014
2015
2016
2017
2018
2019
Aug
Feb
Oct
Apr
Sep
Feb
Jun
Nov
May
Oct
Mar
Jun
Nov
Mar
Jul
Nov
Mar
Jun
Nov
Feb
Aug
Nov
C++14
C++17
C++20
TS bars start and end where work on detailed specification wording starts ("adopt initial working draft") and ends ("send to publication")
Future starts/ends are shaded to indicate that dates, and TS branches are approximate and subject to change
File System
Networking
Reflection
Lib Fundamentals 1
Lib Fundamentals 2
Lib Fundamentals 3
Parallelism 1
Parallelism 2
Concepts
Ranges
Executors
Tx Memory (maybe merge)
Modules
Concurrency 2
Concurrency 1
Coroutines
2D Graphics

STEP 9

Essentially, computers work and understand only binary, ones and zeros, or Machine Language. Programming in binary isn't effective for human beings. For example, to output the words "Hello World!" to the screen in C++ would appear in binary as:

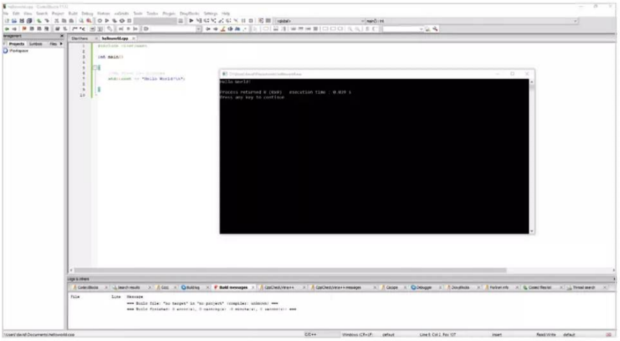
```
01100011 01101111 01110101 01110100 00100000
00111100 00111100 00100000 00100010 01001000
01100101 01101100 01101100 01101111 00100000
01010111 01101111 01110010 01101100 01100100
00100001 00100010 00111011
```



```
01100011 01101111 01110101 01110100 00100000 00111100
00111100 00100000 00100010 01001000 01100101 01101100
01101100 01101111 00100000 01010111 01101111 01110010
01101100 01100100 00100001 00100010 00111011
```

STEP 10

The compiler therefore takes what you've entered as C++ code and translates that to Machine Language. To execute C++ code the IDE 'builds' the code, checking for errors, then pass it through the compiler to check standardisation and convert it to ones and zeros for the computer to act upon. It's rather clever stuff, when you stop to think about it.



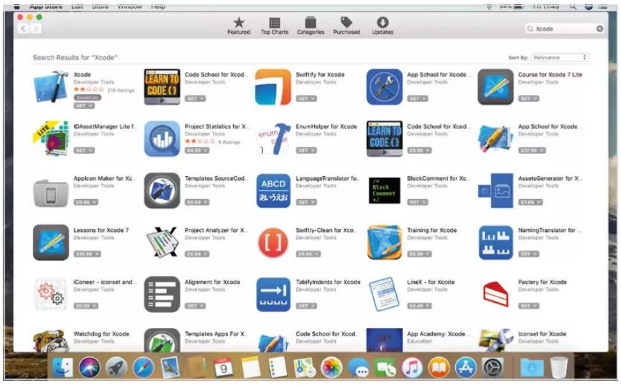
How to Set Up C++ on a Mac

To begin C++ coding on a Mac you first need to install Apple's Xcode. This is a free, full featured IDE that's designed to create native Apple apps. However, you can also use it to create C++ code relatively easily.

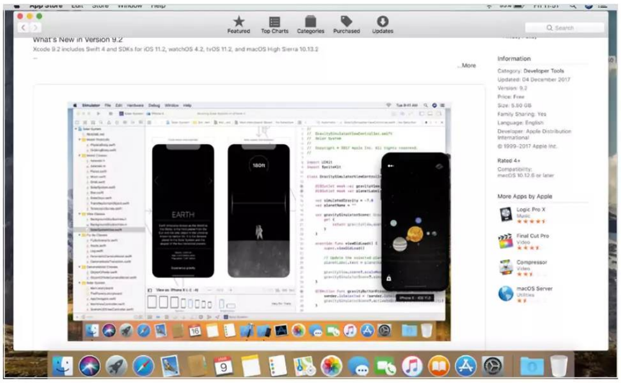
XCODE

Apple's Xcode is primarily designed for users to develop apps for macOS, iOS, tvOS and watchOS applications in Swift or Objective-C but you can use it for C++ too.

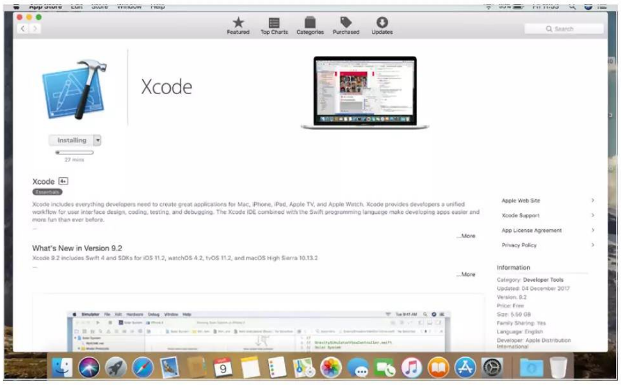
STEP 1 Start by opening the App Store on your Mac, Apple Menu > App Store. In the Search box enter 'Xcode' and press Return. There are many suggestions filling the App Store window but it's the first option, Xcode, that you need to click on.



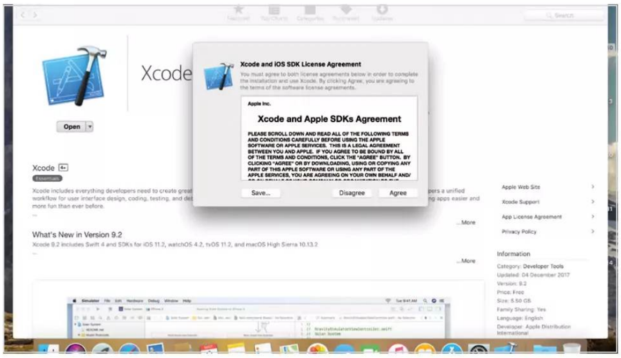
STEP 2 Take a moment to browse through the app's information, including the compatibility, to ensure you have the correct version of macOS. Xcode requires macOS 10.12.6 or later to install and work.



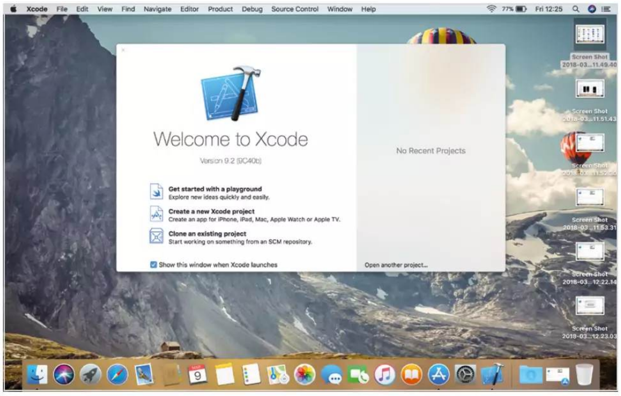
STEP 3 When you're ready, click on the Get button which then turns into Install App. Enter your Apple ID and Xcode begins to download and install. It may take some time depending on the speed of your Internet connection.



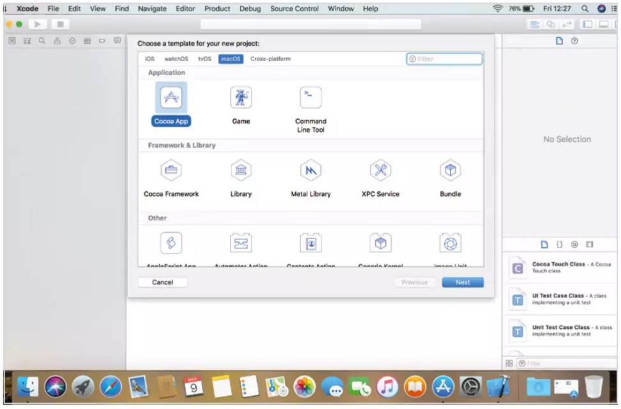
STEP 4 When the installation is complete, click on the Open button to launch Xcode. Click Agree to the licence terms and enter your password to allow Xcode to make changes to the system. When that is done, Xcode begins to install additional components.



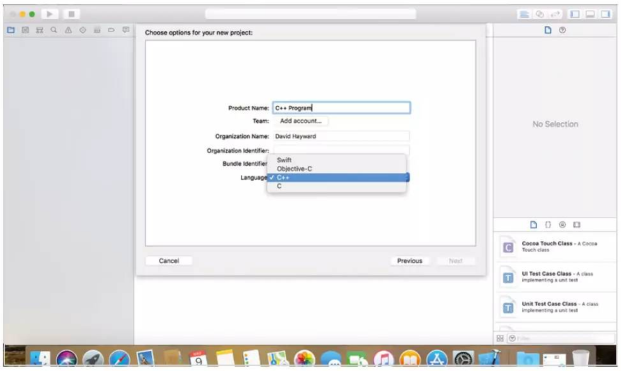
STEP 5 With everything now installed, including the additional components, Xcode launches, displaying the version number along with three choices and any recent projects that you've worked on; with a fresh install though, this is blank.


Welcome to Xcode
Version 9.2 (9C40b)
Get started with a playground
Explore new ideas quickly and easily.
Create a new Xcode project
Create an app for iPhone, iPad, Mac, Apple Watch or Apple TV.
Clone an existing project
Start working on something from an SCM repository.
Show this window when Xcode launches
No Recent Projects
Open another project...

STEP 6 Start by clicking on Create New Xcode Project; this opens a template window to choose which platform you're developing code for. Click the macOS tab, then click the Command Line Tool option. Click Next to continue.


Choose a template for your new project:
iOS
watchOS
tvOS
macOS
Cross-platform
Application
Cocoa App
Game
Command Line Tool
Framework & Library
Cocoa Framework
Library
Metal Library
XPC Service
Bundle
Other
Cancel
Previous
Next
No Selection

STEP 7 Fill in the various fields but ensure that the Language option at the bottom is set to C++; simply choose it from the drop-down list. When you've filled in the fields, and made sure that C++ is the chosen language, click on the Next button to continue.

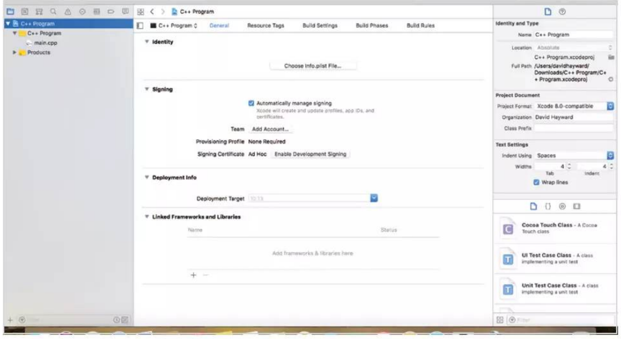


Choose options for your new project:

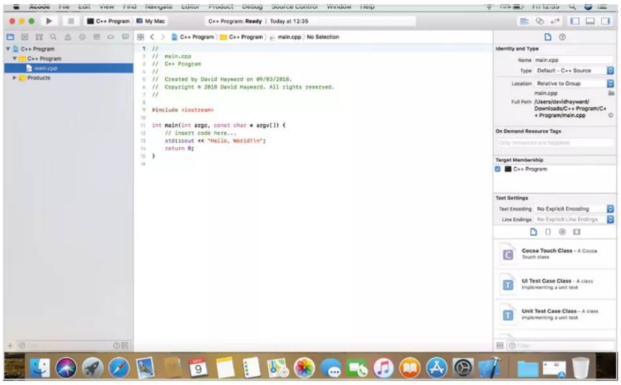
Product Name: C++ Program
Team: Add account...
Organization Name: David Hayward
Organization Identifier:
Bundle Identifier:
Language: C++

Swift
Objective-C
C++
C
Cancel
Previous
Next
No Selection

STEP 8 The next step asks where to create a Git Repository for all your future code. Choose a location on your Mac, or a network location, and click the Create button. When you've done all that, you can start to code. The left-hand pane details the files used in the C++ program you're coding. Click on the main.cpp file in the list.

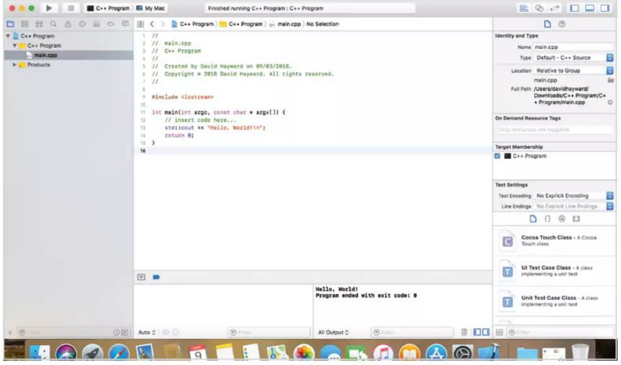

C++ Program
main.cpp
Products
General
Resource Tags
Build Settings
Build Phases
Build Rules
Identity
Signing
Deployment Info
Linked Frameworks and Libraries
Identity and Type
Project Document
Text Settings

STEP 9 You can see that Xcode has automatically completed a basic Hello World program for you. While it may not make much sense at present, you will discover more as you progress, the content is just Xcode utilising what's available on the Mac.



```
//
//  main.cpp
//  C++ Program
//
//  Created by David Hayward on 09/03/2018.
//  Copyright © 2018 David Hayward. All rights reserved.
//

#include <iostream>

int main(int argc, const char * argv[]) {
    // insert code here...
    std::cout << "Hello, World!\n";
    return 0;
}
```

STEP 10 When you want to run the code, click on Product > Run. You may be asked to enable Developer Mode on the Mac; this is to authorise Xcode to perform functions without needing your password every session. When the program executes, the output is displayed at the bottom of the Xcode window.



Finished running C++ Program : C++ Program

```
//
//  main.cpp
//  C++ Program
//
//  Created by David Hayward on 09/03/2018.
//  Copyright © 2018 David Hayward. All rights reserved.
//

#include <iostream>

int main(int argc, const char * argv[]) {
    // insert code here...
    std::cout << "Hello, World!\n";
    return 0;
}
```

```
Hello, World!
Program ended with exit code: 0
```

How to Set Up C++ in Linux

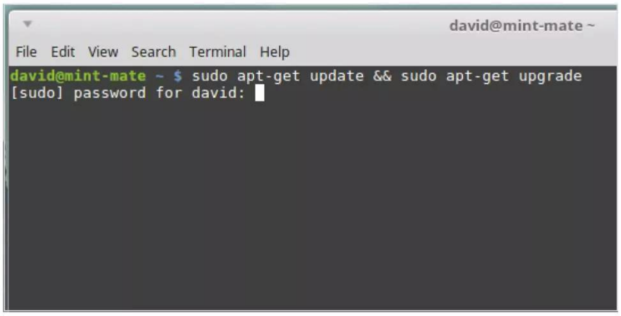
Linux is a great C++ coding environment. Most Linux distros already have the essential components preinstalled, such as a compiler and the text editors are excellent for entering code into, including colour coding; and there's tons of extra software available to help you out.

LINUX++

If you're not familiar with Linux, then we recommend taking a look at one of our Linux titles from the BDM Publications range. If you have a Raspberry Pi, the commands used below work just fine and for this example we're using Linux Mint.

STEP 1

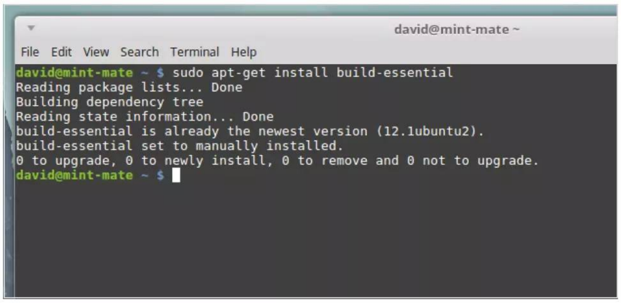
The first step is to ensure Linux is ready for your C++ code, so check the system and software are up to date. Open a Terminal and enter: **sudo apt-get update && sudo apt-get upgrade**. Then press Return and enter your password. These commands update the entire system and any installed software.


david@mint-mate ~
File Edit View Search Terminal Help

```
david@mint-mate ~ $ sudo apt-get update && sudo apt-get upgrade
[sudo] password for david: 
```

STEP 2

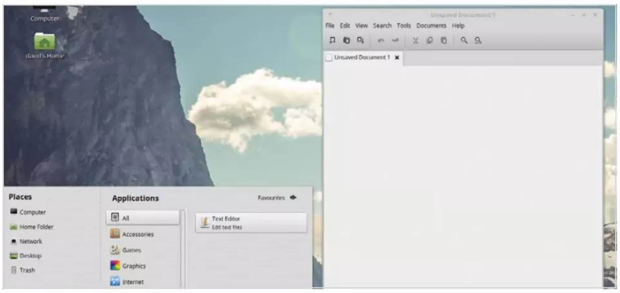
Most Linux distros come preinstalled with all the necessary components to start coding in C++; however, it's always worth checking to see if everything is present. Still within the Terminal, enter: s**udo apt-get install build-essential** and press Return. If you have the right components nothing is installed; if you're missing some then they are installed by the command.


david@mint-mate ~
File Edit View Search Terminal Help

```
david@mint-mate ~ $ sudo apt-get install build-essential
Reading package lists... Done
Building dependency tree
Reading state information... Done
build-essential is already the newest version (12.1ubuntu2).
build-essential set to manually installed.
0 to upgrade, 0 to newly install, 0 to remove and 0 not to upgrade.
david@mint-mate ~ $ 
```

STEP 3

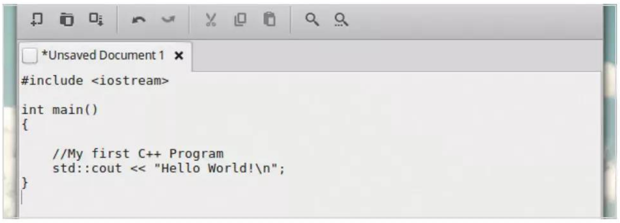
Amazingly, that's it. Everything is already for you to start coding. Here's how to get your first C++ program up and running. In Linux Mint the main text editor is Xed, which you can launch by clicking on the Menu and typing Xed into the search bar. Click on the Text Editor button in the right-hand pane to open it.


Computer
Unsaved Document 1
File Edit View Search Tools Documents Help
Unsaved Document 1
Places
Computer
Home Folder
Network
Desktop
Trash
Applications
Favourites
All
Accessories
Games
Graphics
Internet
Text Editor
Edit text files

STEP 4

In Xed, or any other text editor you may be using, enter the lines of code that make up your C++ Hello World program. It's a little different to what the Mac produced:

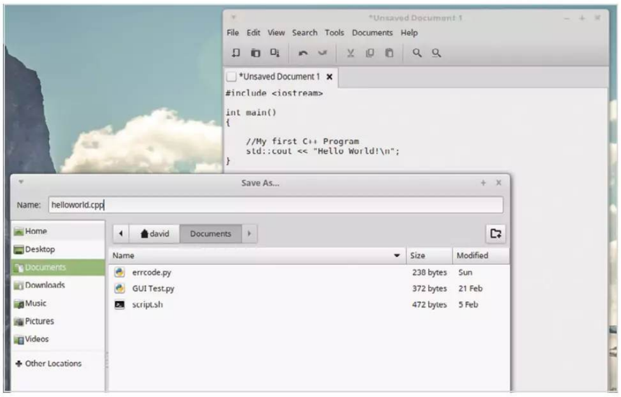
```
#include <iostream>

int main()
{
//My first C++ program
std::cout <<  “Hello World!\n”;
}
```


*Unsaved Document 1

```
#include <iostream>

int main()
{

    //My first C++ Program
    std::cout << "Hello World!\n";
}
```

STEP 5

When you've entered your code, click File > Save As and choose a folder in which to save your program. Name the file as helloworld.cpp (it can be any name as long as it has .cpp as the extension). Click Save to continue.


*Unsaved Document 1
File Edit View Search Tools Documents Help
*Unsaved Document 1
Save As...
Name:
Home
Desktop
Documents
Downloads
Music
Pictures
Videos
Other Locations
david
Documents
Name
Size
Modified

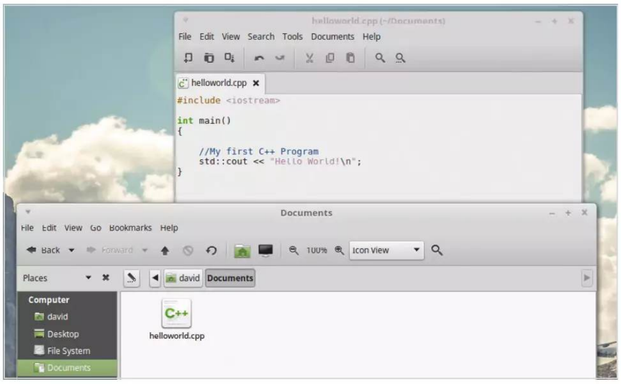
```
#include <iostream>

int main()
{

    //My first C++ Program
    std::cout << "Hello World!\n";
}
```

Name: helloworld.cpp

Name	Size	Modified
errcode.py	238 bytes	Sun
GUI Test.py	372 bytes	21 Feb
script.sh	472 bytes	5 Feb

STEP 6

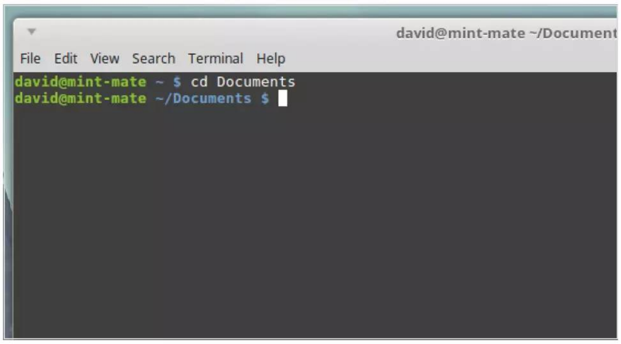
The first thing to notice is that Xed has automatically recognised this as a C++ file, since the file extension is now set to .cpp. The colour coding is present in the code and if you open up the file manager you can also see that file's icon has C++ stamped on it.


helloworld.cpp (~/Documents)
File Edit View Search Tools Documents Help
helloworld.cpp
Documents
File Edit View Go Bookmarks Help
Back
Forward
100%
Icon View
Places
david
Documents
Computer
david
Desktop
File System
Documents
C++
helloworld.cpp

```
#include <iostream>

int main()
{

    //My first C++ Program
    std::cout << "Hello World!\n";
}
```

STEP 7

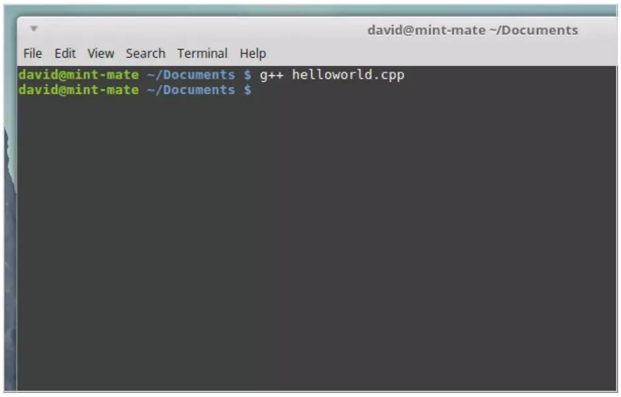
With your code now saved, drop into the Terminal again. You need to navigate to the location of the C++ file you've just saved. Our example is in the Documents folder, so we can navigate to it by entering: **cd Documents**. Remember, the Linux Terminal is case sensitive, so any capitals must be entered correctly.


david@mint-mate ~/Document
File Edit View Search Terminal Help

```
david@mint-mate ~ $ cd Documents
david@mint-mate ~/Documents $ 
```

STEP 8

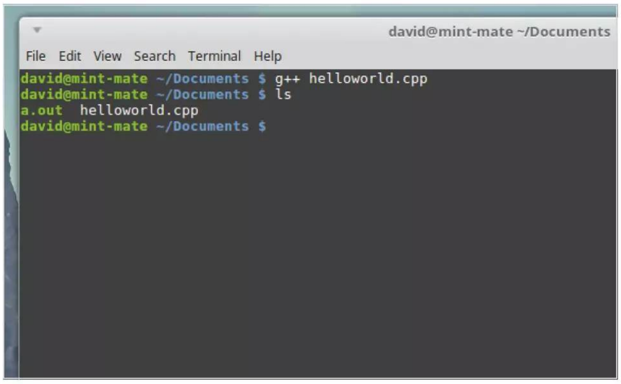
Before you can execute the C++ file you need to compile it. In Linux it's common to use g++, an open source C++ compiler; as you're now in the same folder as the C++ file, enter: **g++ helloworld.cpp** in the Terminal and press Return.


david@mint-mate ~/Documents
File Edit View Search Terminal Help

```
david@mint-mate ~/Documents $ g++ helloworld.cpp
david@mint-mate ~/Documents $
```

STEP 9

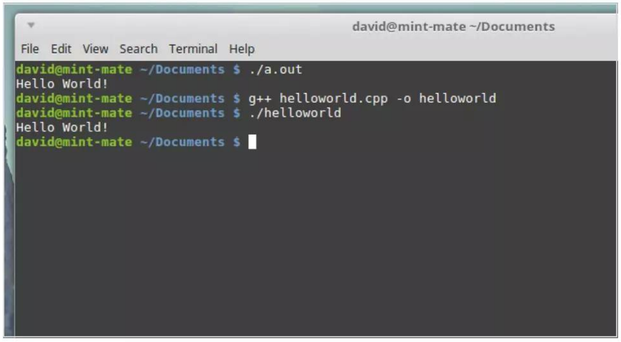
It takes a short time while the code is compiled by g++ but providing there are no mistakes or errors in the code you are returned to the command prompt. The compiling of the code has created a new file. If you enter: ls into the Terminal you can see that alongside your C++ file is a.out.


david@mint-mate ~/Documents
File Edit View Search Terminal Help

```
david@mint-mate ~/Documents $ g++ helloworld.cpp
david@mint-mate ~/Documents $ ls
a.out  helloworld.cpp
david@mint-mate ~/Documents $
```

STEP 10

The a.out file is the compiled C++ code. To run the code enter: **./a.out** and press Return. The words 'Hello World!' appear on the screen. However, a.out isn't very friendly. To name it something else post-compiling, you can recompile with: **g++ helloworld.cpp -o helloworld**. This creates an output file called helloworld which can be run with: **./helloworld**.


david@mint-mate ~/Documents
File Edit View Search Terminal Help

```
david@mint-mate ~/Documents $ ./a.out
Hello World!
david@mint-mate ~/Documents $ g++ helloworld.cpp -o helloworld
david@mint-mate ~/Documents $ ./helloworld
Hello World!
david@mint-mate ~/Documents $ 
```

Other C++ IDEs to Install

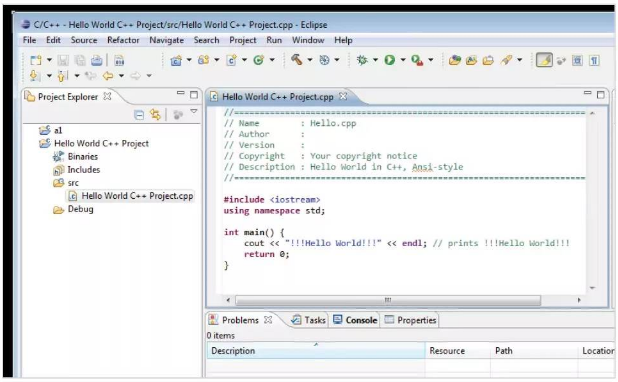
If you want to try a different approach to working with your C++ code, then there are plenty of options available to you. Windows is the most prolific platform for C++ IDEs but there are plenty for Mac and Linux users too.

DEVELOPING C++

Here are ten great C++ IDEs that are worth looking into. You can install one or all of them if you like, but find the one that works best for you.

ECLIPSE

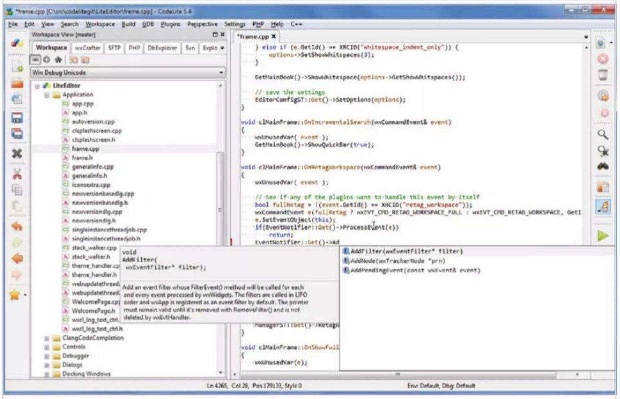
Eclipse is a hugely popular C++ IDE that offers the programmer a wealth of features. It has a great, clean interface, is easy to use and available for Windows, Linux and Mac. Head over to **www.eclipse.org/downloads/** to download the latest version. If you're stuck, click the Need Help link for more information.


C/C++ - Hello World C++ Project/src/Hello World C++ Project.cpp - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
al
Hello World C++ Project
Binaries
Includes
src
Hello World C++ Project.cpp
Debug
Hello World C++ Project.cpp
Problems
Tasks
Console
Properties
0 items
Description
Resource
Path
Location

```
//============================================================================
// Name        : Hello.cpp
// Author      :
// Version     :
// Copyright   : Your copyright notice
// Description : Hello World in C++, Ansi-style
//============================================================================

#include <iostream>
using namespace std;

int main() {
	cout << "!!!Hello World!!!" << endl; // prints !!!Hello World!!!
	return 0;
}
```

CODELITE

CodeLite is a free and open source IDE that's regularly updated and available for Windows, Linux and macOS. It's lightweight, uncomplicated and extremely powerful. You can find out more information as well as how to download and install it at **www.codelite.org/**.


*frame.cpp [C:\src\codelite\git\LiteEditor\frame.cpp] - CodeLite 5.4
File Edit View Search Workspace Build GDB Plugins Perspective Settings PHP Help C++
Workspace View [master]
Workspace
wxCrafter
SFTP
PHP
DbExplorer
Svn
Explo
Win Debug Unicode
LiteEditor
Application
app.cpp
app.h
autoversion.cpp
clsplashscreen.cpp
clsplashscreen.h
frame.cpp
frame.h
generalinfo.cpp
generalinfo.h
iconsextra.cpp
newversiondlg.cpp
newversiondlg.h
newversiondlg.cpp
newversiondlg.h
singleinstancethreadjob.cpp
singleinstancethreadjob.h
stack_walker.cpp
stack_walker.h
theme_handler.cpp
theme_handler.h
webupdatethread.cpp
webupdatethread.h
WelcomePage.cpp
WelcomePage.h
wxcl_log_text_ctrl.cpp
wxcl_log_text_ctrl.h
ClangCodeCompletion
Controls
Debugger
Dialogs
Docking Windows
*frame.cpp
Ln 4265, Col 28, Pos 179133, Style 0
Env: Default, Dbg: Default

```
    } else if (e.GetId() == XRCID("whitespace_indent_only")) {
        options->SetShowWhitspaces(3);
    }

    GetMainBook()->ShowWhitespace(options->GetShowWhitspaces());

    // save the settings
    EditorConfigST::Get()->SetOptions(options);
}

void clMainFrame::OnIncrementalSearch(wxCommandEvent& event)
{
    wxUnusedVar( event );
    GetMainBook()->ShowQuickBar(true);
}

void clMainFrame::OnRetagWorkspace(wxCommandEvent& event)
{
    wxUnusedVar( event );

    // See if any of the plugins want to handle this event by itself
    bool fullRetag = !(event.GetId() == XRCID("retag_workspace"));
    wxCommandEvent e(fullRetag ? wxEVT_CMD_RETAG_WORKSPACE_FULL : wxEVT_CMD_RETAG_WORKSPACE, GetI
    e.SetEventObject(this);
    if(EventNotifier::Get()->ProcessEvent(e))
        return;
    EventNotifier::Get()->Ad
```

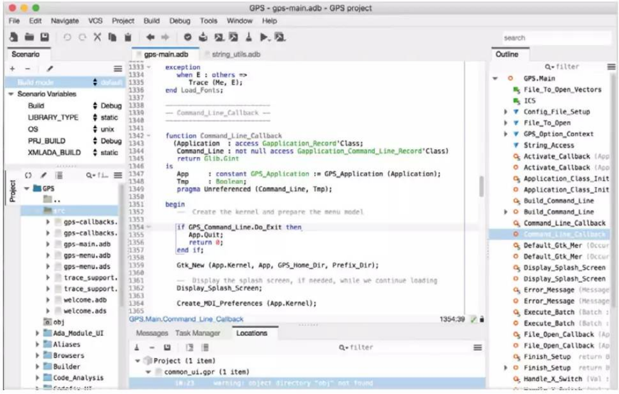
```
void
AddFilter(
wxEventFilter* filter);
```

```
AddFilter(wxEventFilter* filter)
AddNode(wxTrackerNode *prn)
AddPendingEvent(const wxEvent& event)
```

Add an event filter whose FilterEvent() method will be called for each and every event processed by wxWidgets. The filters are called in LIFO order and wxApp is registered as an event filter by default. The pointer must remain valid until it's removed with RemoveFilter() and is not deleted by wxEvtHandler.

GNAT

The GNAT Programming Studio (GPS) is a powerful and intuitive IDE that supports testing, debugging and code analysis. The Community Edition is free, whereas the Pro version costs; however, the Community Edition is available for Windows, Mac, Linux and even the Raspberry Pi. You can find it at **www.adacore.com/download**.


GPS - gps-main.adb - GPS project
File Edit Navigate VCS Project Build Debug Tools Window Help
search
Scenario
Build mode
default
Scenario Variables
Project
GPS
obj
Ada_Module_UI
Aliases
Browsers
Builder
Code_Analysis
gps-callbacks.
gps-callbacks.
gps-main.adb
gps-menu.adb
gps-menu.ads
trace_support.
trace_support.
welcome.adb
welcome.ads
gps-main.adb
string_utils.adb
GPS.Main.Command_Line_Callback
1354:39
Messages
Task Manager
Locations
filter
Outline
filter
GPS.Main
File_To_Open_Vectors
ICS
Config_File_Setup
File_To_Open
GPS_Option_Context
String_Access
Activate_Callback
Activate_Callback
Application_Class_Init
Application_Class_Init
Build_Command_Line
Build_Command_Line
Command_Line_Callback
Command_Line_Callback
Default_Gtk_Mer
Default_Gtk_Mer
Display_Splash_Screen
Display_Splash_Screen
Error_Message
Error_Message
Execute_Batch
Execute_Batch
File_Open_Callback
File_Open_Callback
Finish_Setup
Finish_Setup
Handle_X_Switch

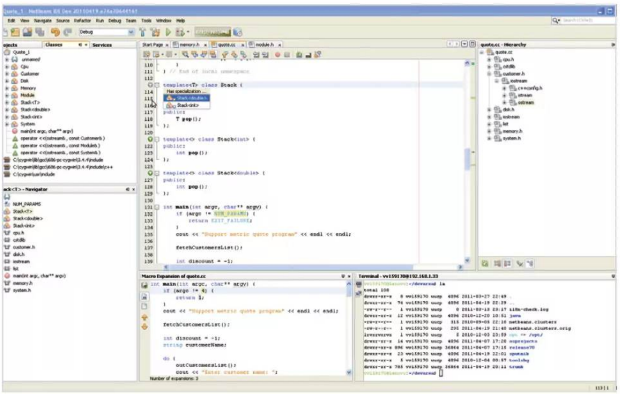
Scenario Variables	
Build	Debug
LIBRARY_TYPE	static
OS	unix
PRJ_BUILD	Debug
XMLADA_BUILD	static

```
   exception
      when E : others =>
         Trace (Me, E);
   end Load_Fonts;

   --------------------------
   -- Command_Line_Callback --
   --------------------------

   function Command_Line_Callback
     (Application  : access Gapplication_Record'Class;
      Command_Line : not null access Gapplication_Command_Line_Record'Class)
      return Glib.Gint
   is
      App     : constant GPS_Application := GPS_Application (Application);
      Tmp     : Boolean;
      pragma Unreferenced (Command_Line, Tmp);

   begin
      --  Create the kernel and prepare the menu model

      if GPS_Command_Line.Do_Exit then
         App.Quit;
         return 0;
      end if;

      Gtk_New (App.Kernel, App, GPS_Home_Dir, Prefix_Dir);

      --  Display the splash screen, if needed, while we continue loading
      Display_Splash_Screen;

      Create_MDI_Preferences (App.Kernel);
```

```
Project (1 item)
  common_ui.gpr (1 item)
    18:23    warning: object directory "obj" not found
```

NETBEANS

Another popular choice is NetBeans. This is another excellent IDE that's packed with features and a pleasure to use. NetBeans IDE includes project based templates for C++ that give you the ability to build applications with dynamic and static libraries. Find out more at **www.netbeans.org/features/cpp/index.html.**

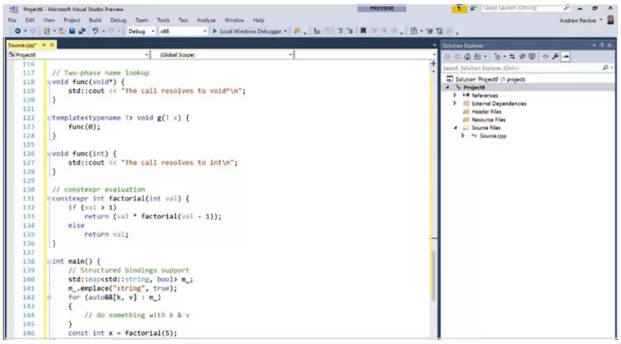

File Edit View Navigate Source Refactor Run Debug Team Tools Window Help
Debug
Projects
Classes
Services
Start Page
memory.h
quote.cc
module.h
quote.cc - Hierarchy
Has specialization ...
Stack<double>
Stack<int>
Macro Expansion of quote.cc
Terminal - vv159170@192.168.1.33
Navigator

```
    }
} // End of local namespace

template<T> class Stack {
public:
    T pop();
};

template<> class Stack<int> {
public:
    int pop();
};

template<> class Stack<double> {
public:
    int pop();
};

int main(int argc, char** argv) {
    if (argc != NUM_PARAMS) {
        return EXIT_FAILURE;
    }
    cout << "Support metric quote program" << endl << endl;

    fetchCustomersList();

    int discount = -1;
```

```
int main(int argc, char** argv) {
    if (argc != 4) {
        return 1;
    }
    cout << "Support metric quote program" << endl << endl;

    fetchCustomersList();

    int discount = -1;
    string customerName;

    do {
        outCustomersList();
```

VISUAL STUDIO

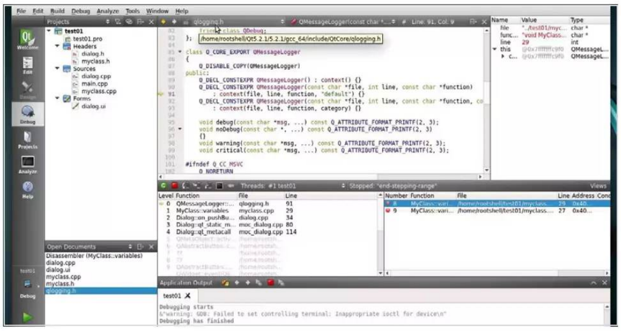
Microsoft's Visual Studio is a mammoth C++ IDE that allows you to create applications for Windows, Android, iOS and the web. The Community version is free to download and install but the other versions allow a free trial period. Go to **www.visualstudio.com/** to see what it can do for you.



```
// Two-phase name lookup
void func(void*) {
    std::cout << "The call resolves to void*\n";
}

template<typename T> void g(T x) {
    func(0);
}

void func(int) {
    std::cout << "The call resolves to int\n";
}

// constexpr evaluation
constexpr int factorial(int val) {
    if (val > 1)
        return (val * factorial(val - 1));
    else
        return val;
}

int main() {
    // Structured bindings support
    std::map<std::string, bool> m_;
    m_.emplace("string", true);
    for (auto&&[k, v] : m_)
    {
        // do something with k & v
    }
    const int x = factorial(5);
```

QT CREATOR

This cross-platform IDE is designed to create C++ applications for desktop and mobile environments. It comes with a code editor and integrated tools for testing and debugging, as well as deploying to you chosen platform. It's not free but there is a trial period on offer before requiring purchasing: **www.qt.io/qt-features-libraries-apis-tools-and-ide/**.

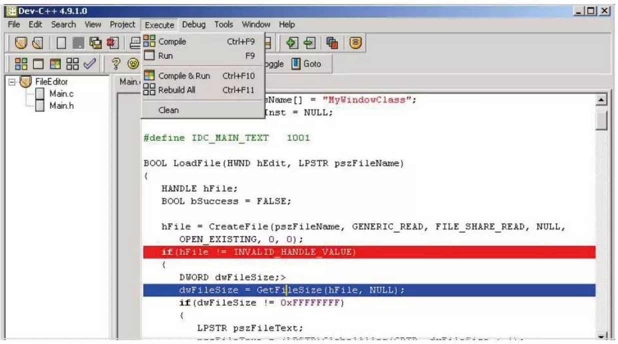


```
    friend class QDebug;
};

class Q_CORE_EXPORT QMessageLogger
{
    Q_DISABLE_COPY(QMessageLogger)
public:
    Q_DECL_CONSTEXPR QMessageLogger() : context() {}
    Q_DECL_CONSTEXPR QMessageLogger(const char *file, int line, const char *function)
        : context(file, line, function, "default") {}
    Q_DECL_CONSTEXPR QMessageLogger(const char *file, int line, const char *function, co
        : context(file, line, function, category) {}

    void debug(const char *msg, ...) const Q_ATTRIBUTE_FORMAT_PRINTF(2, 3);
    void noDebug(const char *, ...) const Q_ATTRIBUTE_FORMAT_PRINTF(2, 3)
    {}
    void warning(const char *msg, ...) const Q_ATTRIBUTE_FORMAT_PRINTF(2, 3);
    void critical(const char *msg, ...) const Q_ATTRIBUTE_FORMAT_PRINTF(2, 3);

#ifndef Q_CC_MSVC
    Q_NORETURN
```

```
Debugging starts
&"warning: GDB: Failed to set controlling terminal: Inappropriate ioctl for device\n"
Debugging has finished
```

DEV C++

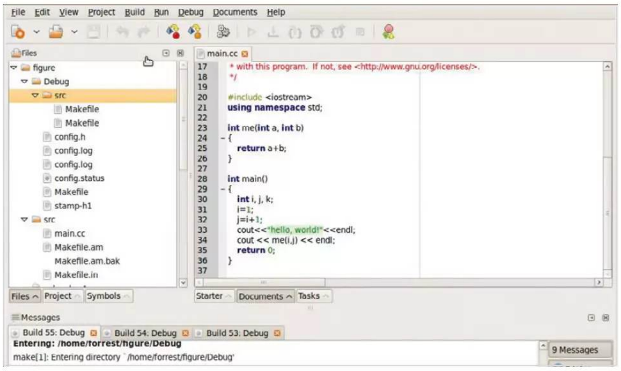
Bloodshed Dev C++, despite its colourful name, is an older IDE that is for Windows systems only. However, many users praise its clean interface and uncomplicated way of coding and compiling. Although there's not been much updating for some time, it's certainly one to consider if you want something different: **www.bloodshed.net/devcpp.html**.



```
sName[] = "MyWindowClass";
Inst = NULL;

#define IDC_MAIN_TEXT   1001

BOOL LoadFile(HWND hEdit, LPSTR pszFileName)
{
   HANDLE hFile;
   BOOL bSuccess = FALSE;

   hFile = CreateFile(pszFileName, GENERIC_READ, FILE_SHARE_READ, NULL,
      OPEN_EXISTING, 0, 0);
   if(hFile != INVALID_HANDLE_VALUE)
   {
      DWORD dwFileSize;>
      dwFileSize = GetFileSize(hFile, NULL);
      if(dwFileSize != 0xFFFFFFFF)
      {
         LPSTR pszFileText;
```

ANJUTA

The Anjuta DevStudio is a Linux-only IDE that features some of the more advanced features you would normally find in a paid software development studio. There's a GUI designer, source editor, app wizard, interactive debugger and much more. Go to **www.anjuta.org/** for more information.

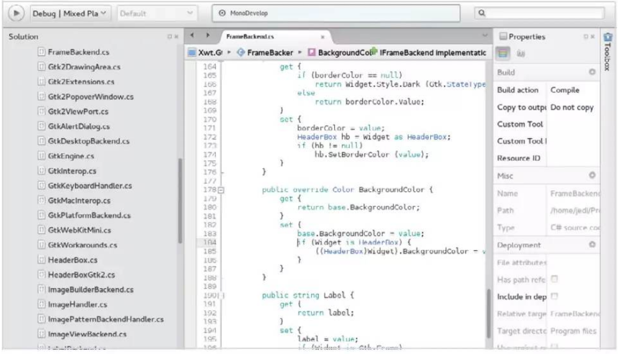


```
 * with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include <iostream>
using namespace std;

int me(int a, int b)
{
   return a+b;
}

int main()
{
   int i, j, k;
   i=1;
   j=i+1;
   cout<<"hello, world!"<<endl;
   cout << me(i,j) << endl;
   return 0;
}
```

```
Entering: /home/forrest/figure/Debug
make[1]: Entering directory `/home/forrest/figure/Debug'
```

MONODEVELOP

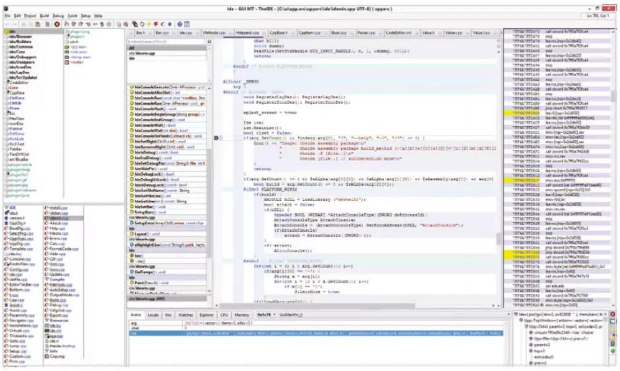
This excellent IDE allows developers to write C++ code for desktop and web applications across all the major platforms. There's an advanced text editor, integrated debugger and a configurable workbench to help you create your code. It's available for Windows, Mac and Linux and is free to download and use: **www.monodevelop.com/**.



```
        get {
            if (borderColor == null)
                return Widget.Style.Dark (Gtk.StateType
            else
                return borderColor.Value;
        }
        set {
            borderColor = value;
            HeaderBox hb = Widget as HeaderBox;
            if (hb != null)
                hb.SetBorderColor (value);
        }
    }

    public override Color BackgroundColor {
        get {
            return base.BackgroundColor;
        }
        set {
            base.BackgroundColor = value;
            if (Widget is HeaderBox) {
                ((HeaderBox)Widget).BackgroundColor = v
            }
        }
    }

    public string Label {
        get {
            return label;
        }
        set {
            label = value;
```

U++

Ultimate++ is a cross-platform C++ IDE that boats a rapid development of code through the smart and aggressive use of C++. For the novice, it's a beast of an IDE but behind its complexity is a beauty that would make a developer's knees go wobbly. Find out more at **www.ultimatepp.org/index.html**.



C++ Fundamentals

Within this section you can begin to understand the structure of C++ code and how to compile and execute that code. These are the fundamentals of C++, which teach you the basics such as using comments, variables, data types, strings and how to use C++ mathematics.

These are the building blocks of a C++ program. With them, you can form your own code, produce an output to the screen and store and retrieve data.

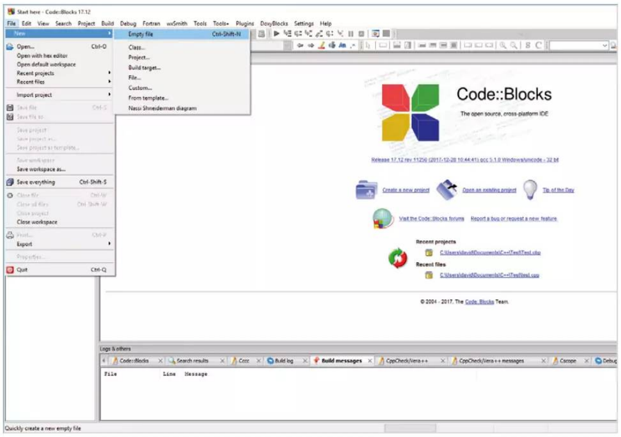
Your First C++ Program

You may have followed the Mac and Linux examples previously but you're going to be working exclusively in Windows and Code::Blocks from here on. Let's begin by writing your first C++ program and taking the first small step into a larger coding world.

HELLO, WORLD!

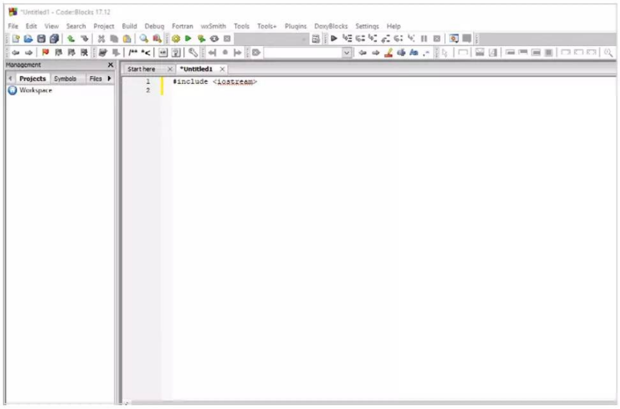
It's traditional in programming for the first code to be entered to output the words 'Hello, World!' to the screen. Interestingly, this dates back to 1968 using a language called BCPL.

STEP 1 As mentioned, we're using Windows 10 and the latest version of Code::Blocks for the rest of the C++ code in this book. Begin by launching Code::Blocks. When open, click on File > New > Empty File or press Ctrl+Shift+N on the keyboard.


Start here - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
New
Open... Ctrl-O
Open with hex editor
Open default workspace
Recent projects
Recent files
Import project
Save file Ctrl-S
Save file as...
Save project
Save project as...
Save project as template...
Save workspace
Save workspace as...
Save everything Ctrl-Shift-S
Close file Ctrl-W
Close all files Ctrl-Shift-W
Close project
Close workspace
Print... Ctrl-P
Export
Properties...
Quit Ctrl-Q
Empty file Ctrl-Shift-N
Class...
Project...
Build target...
File...
Custom...
From template...
Nassi Shneiderman diagram
Code::Blocks
The open source, cross-platform IDE
Create a new project
Open an existing project
Tip of the Day
Visit the Code::Blocks forums
Report a bug or request a new feature
Recent projects
Recent files
Logs & others
Code::Blocks
Search results
Cccc
Build log
Build messages
CppCheck/Vera++
CppCheck/Vera++ messages
Cscope
File
Line
Message
Quickly create a new empty file

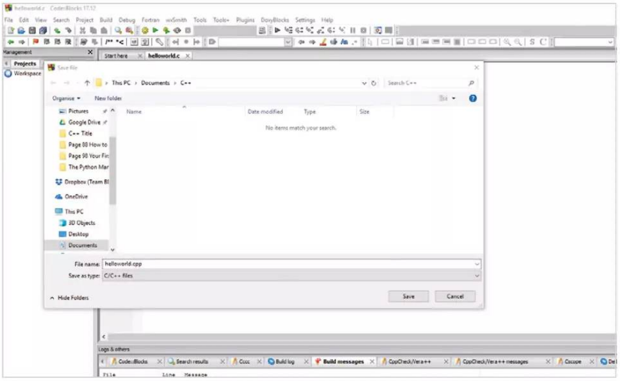
STEP 2 Now you can see a blank screen, with the tab labelled *Untitled1, and the number one in the top left of the main Code::Blocks window. Begin by clicking in the main window, so the cursor is next to the number one, and entering:

```
#include <iostream>
```


*Untitled1 - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects
Symbols
Files
Workspace
Start here
*Untitled1

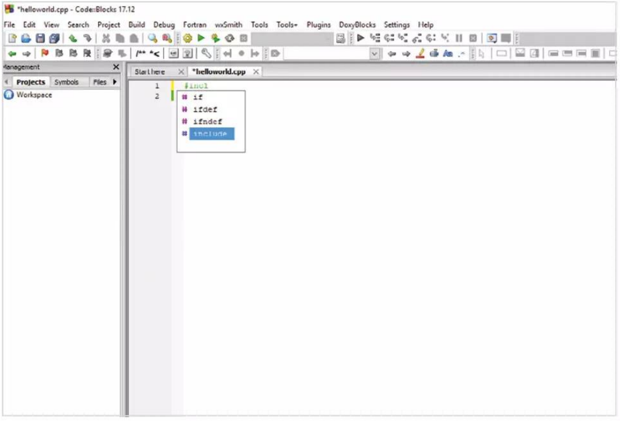
```
#include <iostream>
```

STEP 3 At the moment it doesn't look like much, and it makes even less sense, but we'll get to that in due course. Now click on File > Save File As. Create or find a suitable location on your hard drive and in the File Name box, call it helloworld.cpp. Click the Save as type box and select C/C++ files. Click the Save button.


helloworld.c - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects
Workspace
Start here
helloworld.c
Save File
This PC > Documents > C++
Search C++
Organize
New folder
Pictures
Google Drive
C++ Title
This PC
3D Objects
Desktop
Documents
Name
Date modified
Type
Size
No items match your search.
Save
Cancel
Hide Folders
Logs & others

File name: helloworld.cpp
Save as type: C/C++ files

STEP 4 You can see that Code::Blocks has now changed the colour coding, recognising that the file is now C++ code. This means that code can be auto-selected from the Code::Blocks repository. Delete the #include <iostream> line and re-enter it. You can see the auto-select boxes appearing.


*helloworld.cpp - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects
Symbols
Files
Workspace
Start here
*helloworld.cpp

```
#incl
```

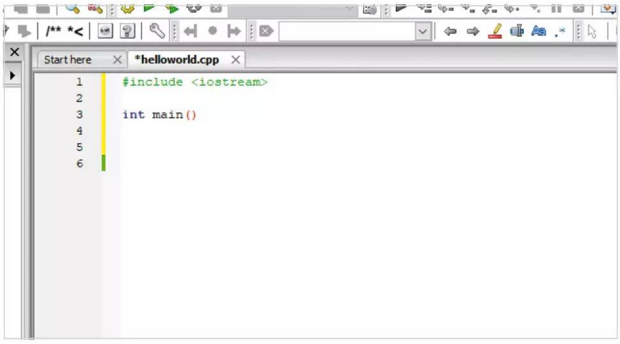
```
if
ifdef
ifndef
include
```

STEP 5

Auto-selection of commands is extremely handy and cuts out potential mistyping. Press Return to get to line 3, then enter:

```
int main()
```

Note: there's no space between the brackets.


Start here
*helloworld.cpp

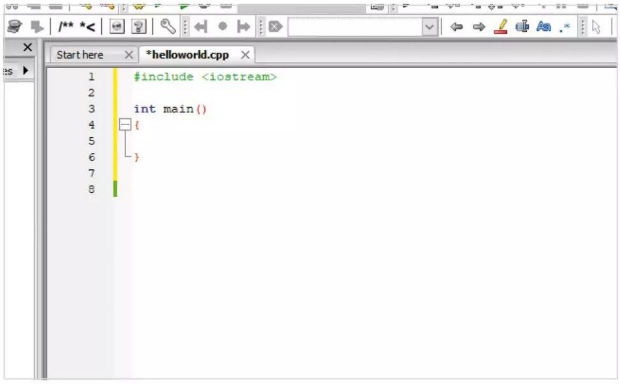
```
1  #include <iostream>
2
3  int main()
4
5
6
```

STEP 6

On the next line below int main(), enter a curly bracket:

```
{
```

This can be done by pressing Shift and the key to the right of P on an English UK keyboard layout.

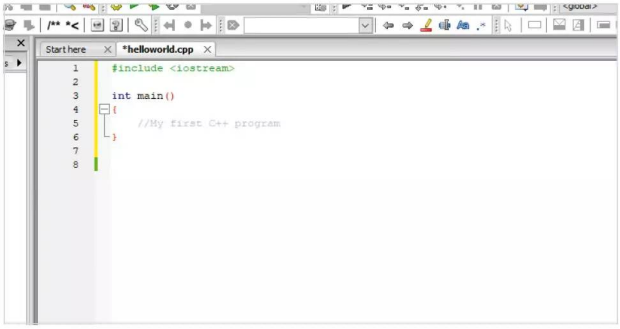

Start here
*helloworld.cpp

```
1  #include <iostream>
2
3  int main()
4  {
5
6  }
7
8
```

STEP 7

Notice that Code::Blocks has automatically created a corresponding closing curly bracket a couple of lines below, linking the pair, as well as a slight indent. This is due to the structure of C++ and it's where the meat of the code is entered. Now enter:

```
//My first C++ program
```

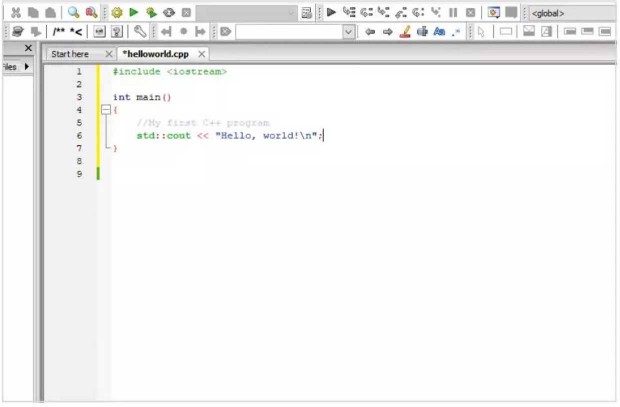

Start here
*helloworld.cpp

```
1  #include <iostream>
2
3  int main()
4  {
5      //My first C++ program
6  }
7
8
```

STEP 8

Note again the colour coding change. Press Return at the end of the previous step's line, and then enter:

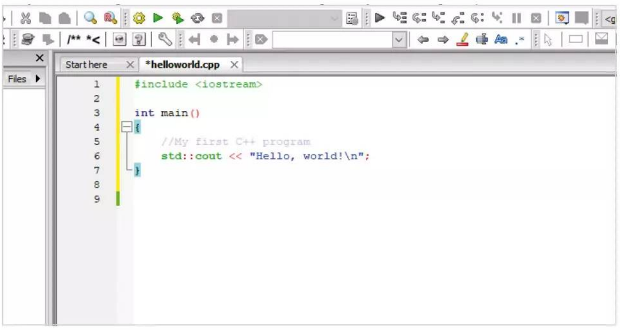
```
std::cout << “Hello, world!\n”;
```


Start here
*helloworld.cpp

```
1  #include <iostream>
2
3  int main()
4  {
5      //My first C++ program
6      std::cout << "Hello, world!\n";
7  }
8
9
```

STEP 9

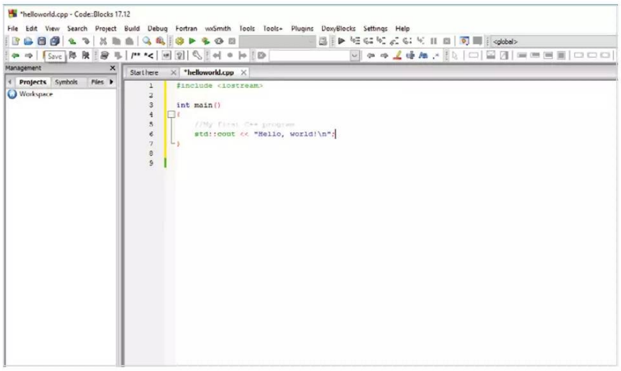
Just as before, Code::Blocks auto-completes the code you're entering, including placing a closing speech mark as soon as you enter the first. Don't forget the semicolon at the end of the line; this is one of the most important elements to a C++ program and we'll tell you why in the next section. For now, move the cursor down to the closing curly bracket and press Return.


Files
Start here
*helloworld.cpp

```
1  #include <iostream>
2
3  int main()
4  {
5      //My first C++ program
6      std::cout << "Hello, world!\n";
7  }
8
9
```

STEP 10

That's all you need to do for the moment. It may not look terribly amazing but C++ is best absorbed in small chunks. Don't execute the code at the moment as you need to look at how a C++ program is structured first; then you can build and run the code. For now, click on Save, the single floppy disc icon.


*helloworld.cpp - Code::Blocks 17.12
File
Edit
View
Search
Project
Build
Debug
Fortran
wxSmith
Tools
Tools+
Plugins
DoxyBlocks
Settings
Help
Save
Management
Projects
Symbols
Files
Workspace
Start here
*helloworld.cpp

```
1  #include <iostream>
2
3  int main()
4  {
5      //My first C++ program
6      std::cout << "Hello, world!\n";
7  }
8
9
```

Structure of a C++ Program

C++ has a very defined structure and way of doing things. Miss something out, even as small as a semicolon, and your entire program will fail to be compiled and executed. Many a professional programmer has fallen foul of sloppy structure.

#INCLUDE <C++ STRUCTURE>

Learning the basics of programming, you begin to understand the structure of a program. The commands may be different from one language to the next, but you will start to see how the code works.

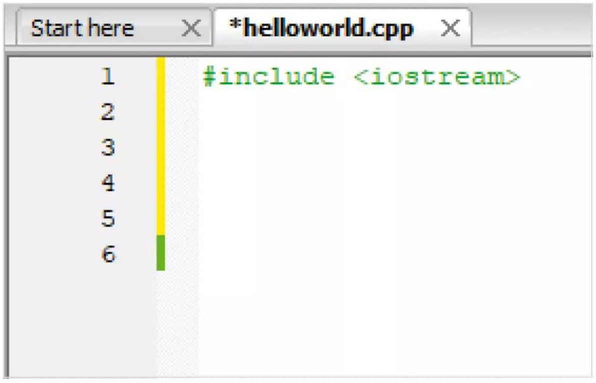
C++

C++ was invented by Danish student Bjarne Stroustrup in 1979, as a part of his Ph.D. thesis. Initially C++ was called C with Classes, which added features to the already popular C programming language, while making it a more user-friendly environment through a new structure.

Bjarne Stroustrup, inventor of C++.

#INCLUDE

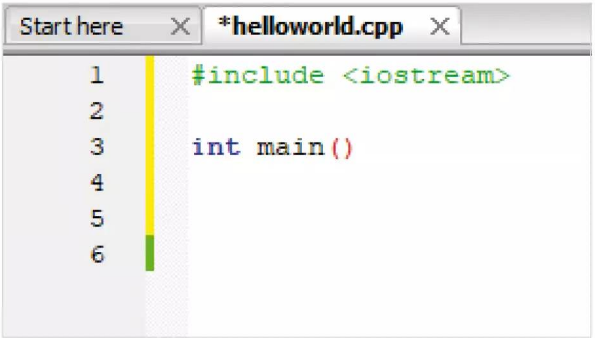
The structure of a C++ program is quite precise. Every C++ code begins with a directive: **#include <>**. The directive instructs the pre-processor to include a section of the standard C++ code. For example: **#include <iostream>** includes the iostream header to support input/output operations.



```
#include <iostream>
```

INT MAIN()

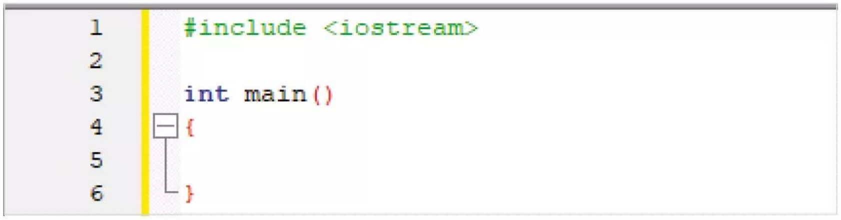
int main() initiates the declaration of a function, which is a group of code statements under the name 'main'. All C++ code begins at the main function, regardless of where it actually lies within the code.



```
#include <iostream>

int main()
```

BRACES

The open brace (curly brackets) is something that you may not have come across before, especially if you're used to Python. The open brace indicates the beginning of the main function and contains all the code that belongs to that function.

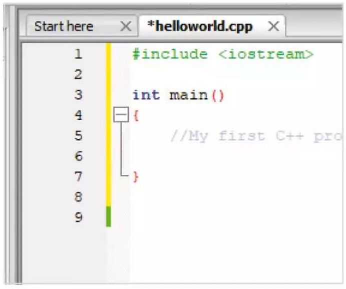


```
#include <iostream>

int main()
{

}
```

COMMENTS

Lines that begin with a double slash are comments. This means they won't be executed in the code and are ignored by the compiler. Comments are designed to help you, or another programmer looking at your code, explain what's going on. There are two types of comment: /* covers multiple line comments, // a single line.

Lines that begin with a double slash are comments. This means they won't be executed in the code and are ignored by the compiler. Comments are designed to help you, or another programmer looking at your code, explain what's going on. There are two types of comment: /* covers multiple line comments, // a single line.

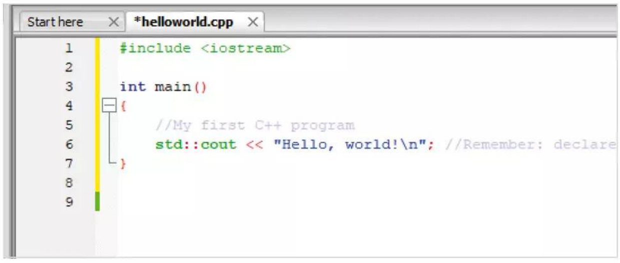

Start here
*helloworld.cpp

```
#include <iostream>

int main()
{
    //My first C++ pro
}

```

STD

While **std** stands for something quite different, in C++ it means Standard. It's part of the Standard Namespace in C++, which covers a number of different statements and commands. You can leave the **std::** part out of the code but it must be declared at the start with: **using namespace std**; not both. For example:

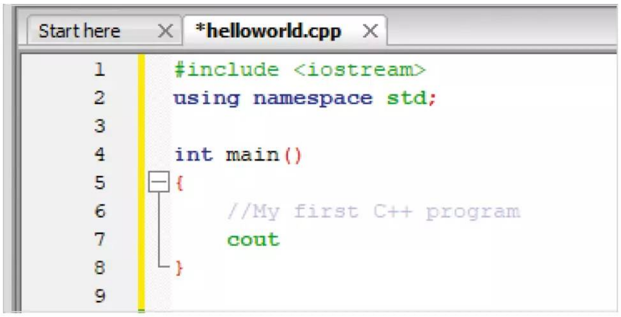
```
#include <iostream>
using namespace std;
```


Start here
*helloworld.cpp

```
#include <iostream>

int main()
{
    //My first C++ program
    std::cout << "Hello, world!\n"; //Remember: declare
}

```

COUT

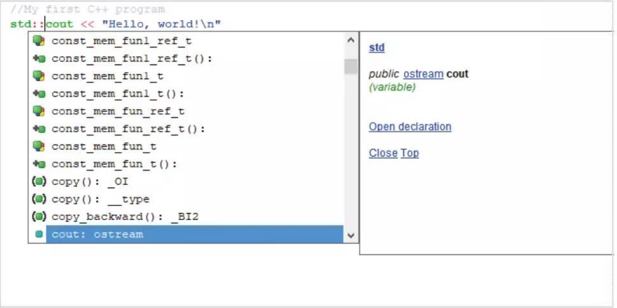
In this example we're using cout, which is a part of the Standard Namespace, hence why it's there, as you're asking C++ to use it from that particular namespace. Cout means Character OUTput, which displays, or prints, something to the screen. If we leave **std::** out we have to declare it at the start of the code, as mentioned previously.


Start here
*helloworld.cpp

```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout
}

```

<<

The two chevrons used here are insertion operators. This means that whatever follows the chevrons is to be inserted into the std::cout statement. In this case they're the words 'Hello, world', which are to be displayed on the screen when you compile and execute the code.

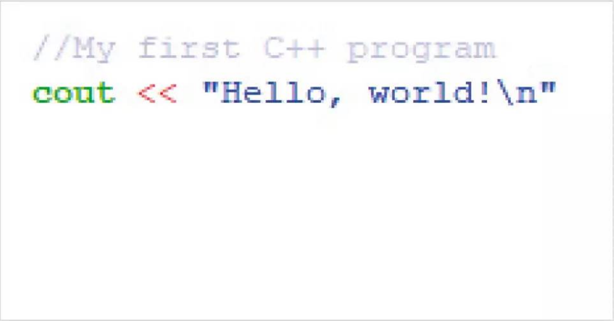


```
//My first C++ program
std::cout << "Hello, world!\n"
```

const_mem_fun1_ref_t
const_mem_fun1_ref_t():
const_mem_fun1_t
const_mem_fun1_t():
const_mem_fun_ref_t
const_mem_fun_ref_t():
const_mem_fun_t
const_mem_fun_t():
copy(): _OI
copy(): __type
copy_backward(): _BI2
cout: ostream
std
public ostream cout
(variable)
Open declaration
Close Top

OUTPUTS

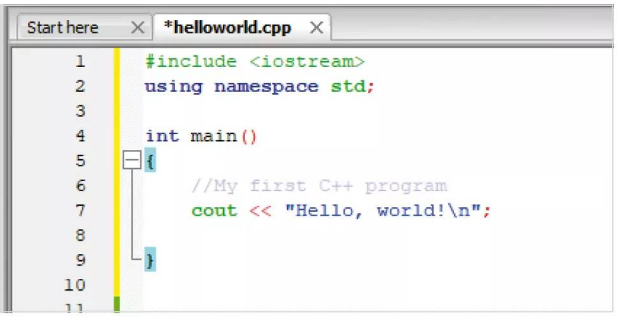
Leading on, the "Hello, world!" part is what we want to appear on the screen when the code is executed. You can enter whatever you like, as long as it's inside the quotation marks. The brackets aren't needed but some compilers insist on them. The \n part indicates a new line is to be inserted.



```
//My first C++ program
cout << "Hello, world!\n"
```

; AND }

Finally you can see that lines within a function code block (except comments) end with a semicolon. This marks the end of the statement and all statements in C++ must have one at the end or the compiler fails to build the code. The very last line has the closing brace to indicate the end of the main function.


Start here
*helloworld.cpp

```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";

}

```

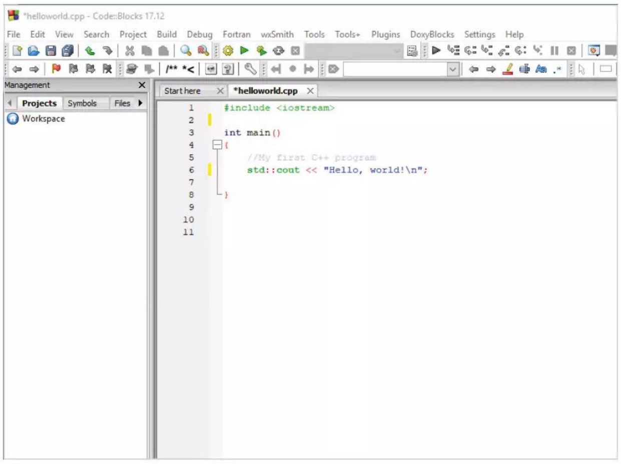
Compile and Execute

You've created your first C++ program and you now understand the basics behind the structure of one. Let's actually get things moving and compile and execute, or run if you prefer, the program and see how it looks.

GREETINGS FROM C++

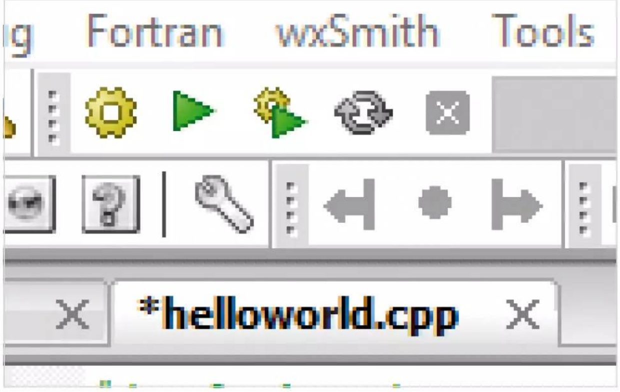
Compiling and executing C++ code from Code::Blocks is extraordinarily easy; just a matter of clicking an icon and seeing the result. Here's how it's done.

STEP 1 Open Code::Blocks, if you haven't already, and load up the previously saved Hello World code you created. Ensure that there are no visible errors, such as missing semicolons at the end of the std::cout line.

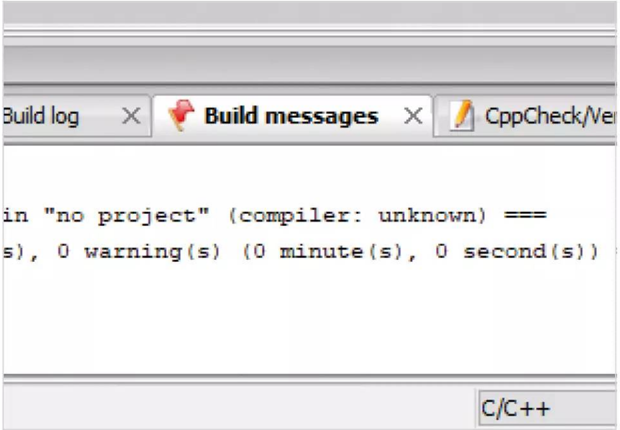

*helloworld.cpp - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects Symbols Files
Workspace
Start here
*helloworld.cpp

```
#include <iostream>

int main()
{
    //My first C++ program
    std::cout << "Hello, world!\n";
}
```

STEP 2 If your code is looking similar to the one in our screenshot, then look to the menu bar along the top of the screen. Under the Fortran entry in the topmost menu you can see a group of icons: a yellow cog, green play button and a cog/play button together. These are Build, Run, Build and Run functions.

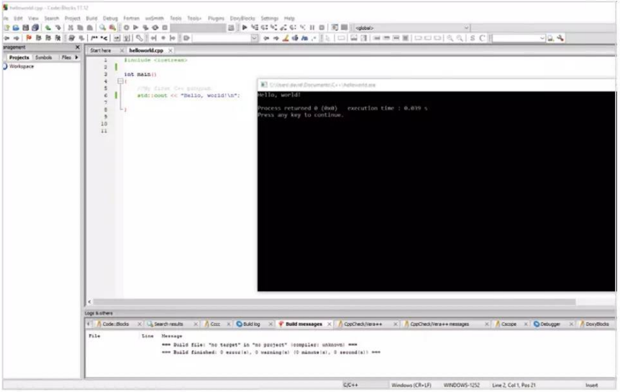

Fortran
wxSmith
Tools
*helloworld.cpp

STEP 3 Start by clicking on the Build icon, the yellow cog. At this point, your code has now been run through the Code::Blocks compiler and checked for any errors. You can see the results of the Build by looking to the bottom window pane. Any messages regarding the quality of the code are displayed here.


Build log
Build messages
CppCheck/Ver
C/C++

```
in "no project" (compiler: unknown) ===
s), 0 warning(s) (0 minute(s), 0 second(s))
```

STEP 4 Now click on the Run icon, the green play button. A command line box appears on your screen displaying the words: Hello, world!, followed by the time it's taken to execute the code, and asking you press a key to continue. Well done, you just compiled and executed your first C++ program.


Start here
helloworld.cpp
Logs & others
Code::Blocks
Search results
Cccc
Build log
Build messages
CppCheck/Vera++
CppCheck/Vera++ messages
Cscope
Debugger
DoxyBlocks
C/C++
Windows (CR+LF)
WINDOWS-1252
Line 2, Col 1, Pos 21
Insert

```
Hello, world!

Process returned 0 (0x0)   execution time : 0.039 s
Press any key to continue.
```

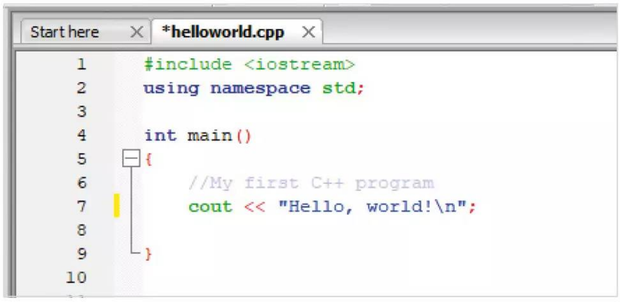
```
File            Line   Message
                       === Build file: "no target" in "no project" (compiler: unknown) ===
                       === Build finished: 0 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

STEP 5 Pressing any key in the command line box closes it, returning you to Code::Blocks. Let's alter the code slightly. Under the #include line, enter:

```
using namespace std;
```

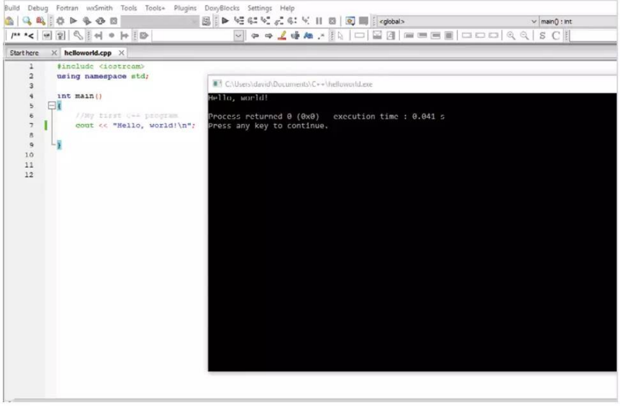
Then, delete the std:: part of the Cout line; like so:

```
cout << “Hello, world\n”;
```



```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
}
```

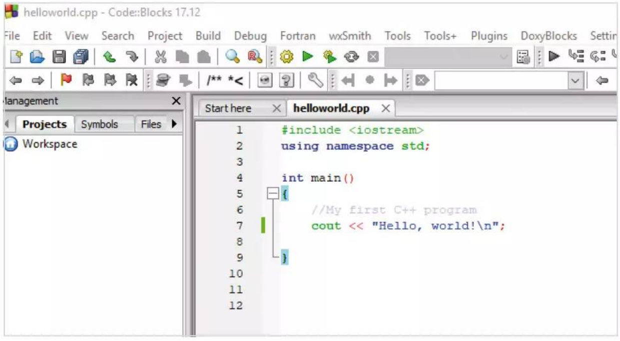
STEP 6 In order to apply the new changes to the code, you need to re-compile, build, and run it again. This time, however, you can simply click the Build/Run icon, the combined yellow cog and green play button.



```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
}
```

```
Hello, world!

Process returned 0 (0x0)   execution time : 0.041 s
Press any key to continue.
```

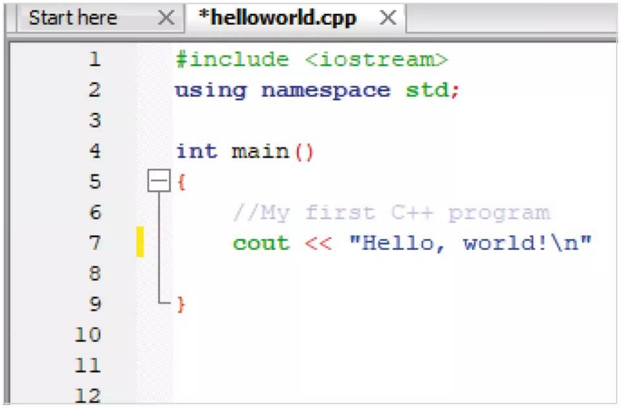
STEP 7 Just as we mentioned in the previous pages, you don't need to have std::cout if you already declare using namespace std; at the beginning of the code. We could have easily clicked the Build/Run icon to begin with but it's worth going through the available options. You can also see that by building and running, the file has been saved.



```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
}
```

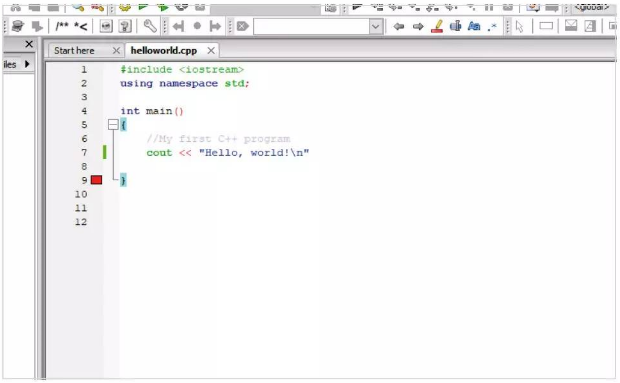
STEP 8 Create a deliberate error in the code. Remove the semicolon from the cout line, so it reads:

```
cout << “Hello, world!\n”
```



```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n"
}
```

STEP 9 Now click the Build and Run icon again to apply the changes to the code. This time Code::Blocks refuses to execute the code, due to the error you put in. In the Log pane at the bottom of the screen you are informed of the error, in this case: Expected ';' before '}' token, indicating the missing semicolon.

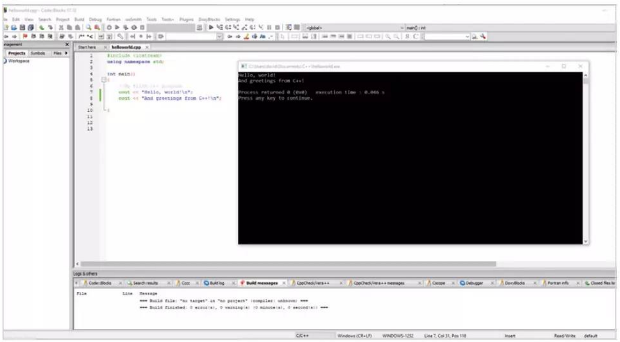


```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n"
}
```

STEP 10 Replace the semicolon and under the cout line, enter a new line to your code:

```
cout << “And greetings from C++!\n”;
```

The \n simply adds a new line under the last line of outputted text. Build and Run the code, to display your handiwork.



```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
    cout << "And greetings from C++!\n";
}
```

```
Hello, world!
And greetings from C++!

Process returned 0 (0x0)   execution time : 0.066 s
Press any key to continue.
```

Using Comments

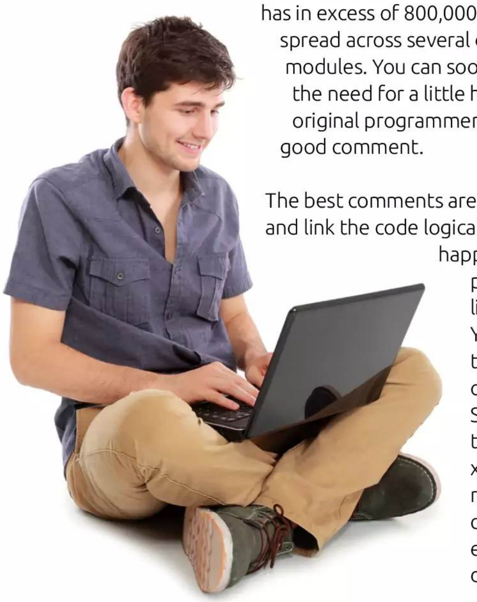
While comments may seem like a minor element to the many lines of code that combine to make a game, application or even an entire operating system, in actual fact they're probably one of the most important factors.

THE IMPORTANCE OF COMMENTING

Comments inside code are basically human readable descriptions that detail what the code is doing at that particular point. They don't sound especially important but code without comments is one of the many frustrating areas of programming, regardless of whether you're a professional or just starting out.

In short, all code should be commented in such a manner as to effectively describe the purpose of a line, section, or individual elements. You should get in to the habit of commenting as much as possible, by imagining that someone who doesn't know anything about programming can pick up your code and understand what it's going to do simply by reading your comments.

In a professional environment, comments are vital to the success of the code and ultimately, the company. In an organisation, many programmers work in teams alongside engineers, other developers, hardware analysts and so on. If you're a part of the team that's writing a bespoke piece of software for the company, then your comments help save a lot of time should something go wrong, and another team member has to pick up and follow the trail to pinpoint the issue.

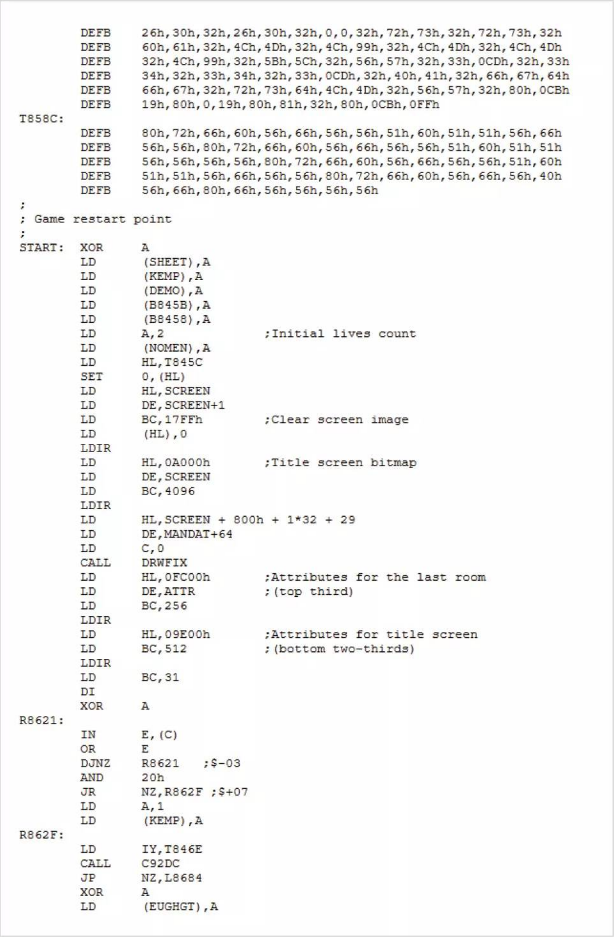


Place yourself in the shoes of someone whose job it is to find out what's wrong with a program. The program has in excess of 800,000 lines of code, spread across several different modules. You can soon appreciate the need for a little help from the original programmers in the form of a good comment.

The best comments are always concise and link the code logically, detailing what happens when the program hits this line or section. You don't need to comment on every line. Something along the lines of: if x==0 doesn't require you to comment that if x equals zero then do something; that's going to be obvious to the reader. However, if x equalling zero is something that drastically changes the program for the user, such as, they've run out of lives, then it certainly needs to commented on.

Even if the code is your own, you should write comments as if you were going to publicly share it with others. This way you can return to that code and always understand what it was you did or where it was you went wrong or what worked brilliantly.

Comments are good practise and once you understand how to add a comment where needed, you soon do it as if it's second nature.



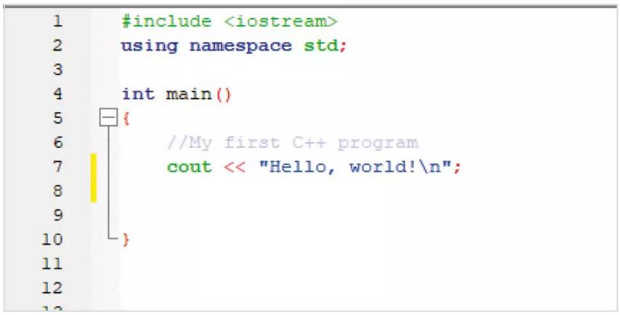
```
        DEFB    26h,30h,32h,26h,30h,32h,0,0,32h,72h,73h,32h,72h,73h,32h
        DEFB    60h,61h,32h,4Ch,4Dh,32h,4Ch,99h,32h,4Ch,4Dh,32h,4Ch,4Dh
        DEFB    32h,4Ch,99h,32h,5Bh,5Ch,32h,56h,57h,32h,33h,0CDh,32h,33h
        DEFB    34h,32h,33h,34h,32h,33h,0CDh,32h,40h,41h,32h,66h,67h,64h
        DEFB    66h,67h,32h,72h,73h,64h,4Ch,4Dh,32h,56h,57h,32h,80h,0CBh
        DEFB    19h,80h,0,19h,80h,81h,32h,80h,0CBh,0FFh
T858C:
        DEFB    80h,72h,66h,60h,56h,66h,56h,56h,51h,60h,51h,51h,56h,66h
        DEFB    56h,56h,80h,72h,66h,60h,56h,66h,56h,56h,51h,60h,51h,51h
        DEFB    56h,56h,56h,56h,80h,72h,66h,60h,56h,66h,56h,56h,51h,60h
        DEFB    51h,51h,56h,66h,56h,56h,80h,72h,66h,60h,56h,66h,56h,40h
        DEFB    56h,66h,80h,66h,56h,56h,56h,56h
;
; Game restart point
;
START:  XOR     A
        LD      (SHEET),A
        LD      (KEMP),A
        LD      (DEMO),A
        LD      (B845B),A
        LD      (B8458),A
        LD      A,2             ;Initial lives count
        LD      (NOMEN),A
        LD      HL,T845C
        SET     0,(HL)
        LD      HL,SCREEN
        LD      DE,SCREEN+1
        LD      BC,17FFh        ;Clear screen image
        LD      (HL),0
        LDIR
        LD      HL,0A000h       ;Title screen bitmap
        LD      DE,SCREEN
        LD      BC,4096
        LDIR
        LD      HL,SCREEN + 800h + 1*32 + 29
        LD      DE,MANDAT+64
        LD      C,0
        CALL    DRWFIX
        LD      HL,0FC00h       ;Attributes for the last room
        LD      DE,ATTR         ;(top third)
        LD      BC,256
        LDIR
        LD      HL,09E00h       ;Attributes for title screen
        LD      BC,512          ;(bottom two-thirds)
        LDIR
        LD      BC,31
        DI
        XOR     A
R8621:
        IN      E,(C)
        OR      E
        DJNZ    R8621   ;$-03
        AND     20h
        JR      NZ,R862F ;$+07
        LD      A,1
        LD      (KEMP),A
R862F:
        LD      IY,T846E
        CALL    C92DC
        JP      NZ,L8684
        XOR     A
        LD      (EUGHGT),A
```

C++ COMMENTS

Commenting in C++ involves using a double forward slash '/', or a forward slash and an asterisk, '/*'. You've already seen some brief examples but this is how they work.

STEP 1 Using the Hello World code as an example, you can easily comment on different sections of the code using the double forward slash:

```
//My first C++ program
cout << “Hello, world!\n”;
```

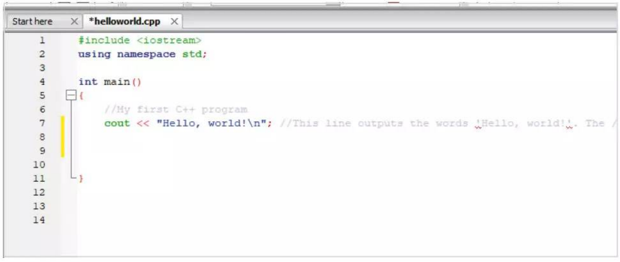


```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";


}
```

STEP 2 However, you can also add comments to the end of a line of code, to describe in a better way what's going on:

```
cout << “Hello, world!\n”; //This line outputs the
words ‘Hello, world!’. The \n denotes a new line.
```

Note, you don't have to put a semicolon at the end of a comment. This is because it's a line in the code that's ignored by the compiler.

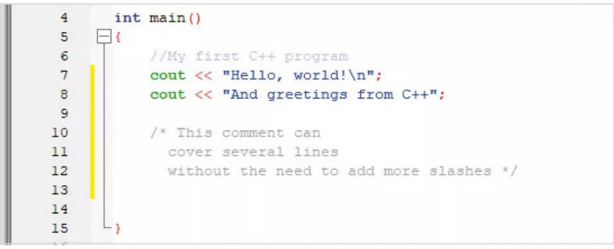


```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n"; //This line outputs the words 'Hello, world!'. The /



}
```

STEP 3 You can comment out several lines by using the forward slash and asterisk:

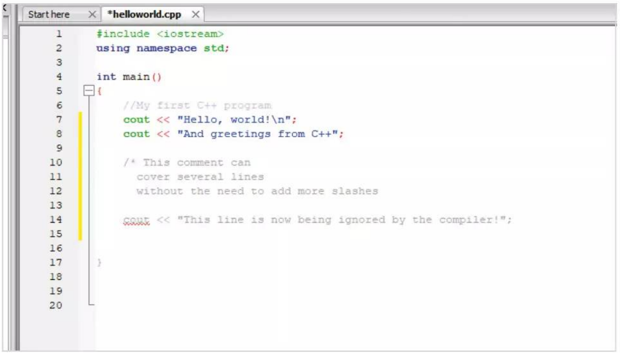
```
/* This comment can
      cover several lines
      without the need to add more slashes */
```

Just remember to finish the block comment with the opposite asterisk and forward slash.



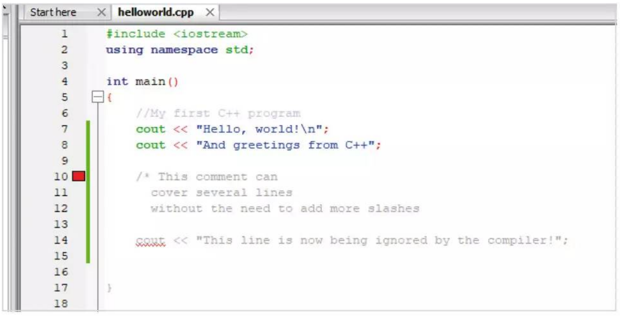
```
int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
    cout << "And greetings from C++";

    /* This comment can
      cover several lines
      without the need to add more slashes */


}
```

STEP 4 Be careful when commenting, especially with block comments. It's very easy to forget to add the closing asterisk and forward slash and thus negate any code that falls inside the comment block.



```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
    cout << "And greetings from C++";

    /* This comment can
      cover several lines
      without the need to add more slashes

    cout << "This line is now being ignored by the compiler!";


}
```

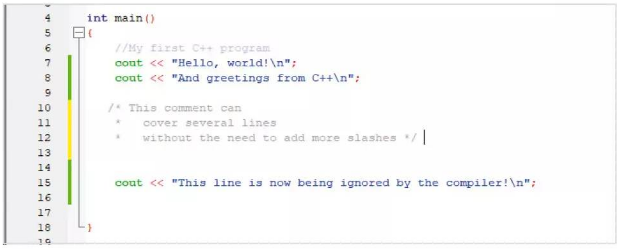
STEP 5 Obviously if you try and build and execute the code it errors out, complaining of a missing curly bracket '}' to finish off the block of code. If you've made the error a few times, then it can be time consuming to go back and rectify. Thankfully, the colour coding in Code::Blocks helps identify comments from code.



```
#include <iostream>
using namespace std;

int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
    cout << "And greetings from C++";

    /* This comment can
      cover several lines
      without the need to add more slashes

    cout << "This line is now being ignored by the compiler!";


}
```

STEP 6 If you're using block comments, it's good practise in C++ to add an asterisk to each new line of the comment block. This also helps you to remember to close the comment block off before continuing with the code:

```
/* This comment can
  *   cover several lines
  *   without the need to add more slashes */
```



```
int main()
{
    //My first C++ program
    cout << "Hello, world!\n";
    cout << "And greetings from C++\n";

   /* This comment can
    *   cover several lines
    *   without the need to add more slashes */


    cout << "This line is now being ignored by the compiler!\n";


}
```

Variables

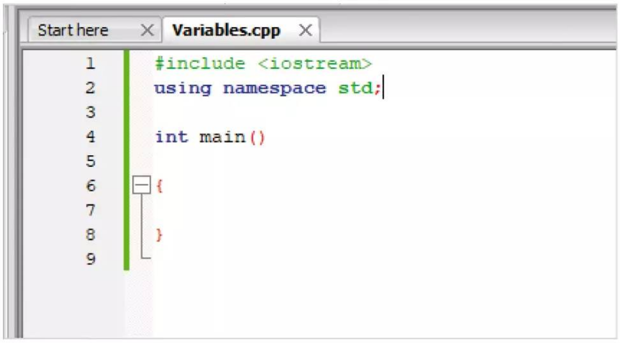
Variables differ slightly when using C++ as opposed to Python. In Python, you can simply state that ‘a’ equals 10 and a variable is assigned. However, in C++ a variable has to be declared with its type before it can be used.

THE DECLARATION OF VARIABLES

You can declare a C++ variable by using statements within the code. There are several distinct types of variables you can declare. Here’s how it works.

STEP 1 Open up a new, blank C++ file and enter the usual code headers:

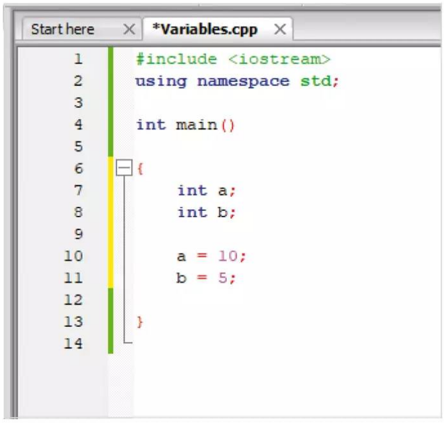
```
#include <iostream>
using namespace std;

int main()
{

}
```



```
#include <iostream>
using namespace std;

int main()

{

}
```

STEP 2 Start simple by creating two variables, a and b, with one having a value of 10 and the other 5. You can use the data type int to declare these variables. Within the curly brackets, enter:

```
int a;
int b;

a = 10;
b = 5;
```

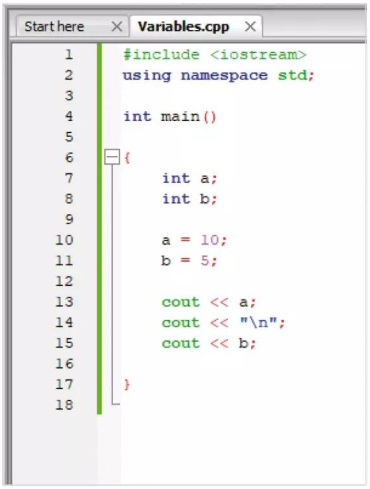


```
#include <iostream>
using namespace std;

int main()

{

    int a;
    int b;

    a = 10;
    b = 5;

}
```

STEP 3 You can build and run the code but it won’t do much, other than store the values 10 and 5 to the integers a and b. To output the contents of the variables, add:

```
cout << a;
cout << ”\n”;
cout << b;
```

The cout << “\n”; part simply places a new line between the output of 10 and 5.

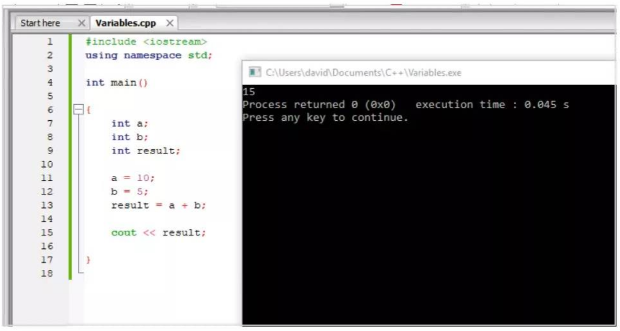


```
#include <iostream>
using namespace std;

int main()

{

    int a;
    int b;

    a = 10;
    b = 5;

    cout << a;
    cout << "\n";
    cout << b;

}
```

STEP 4 Naturally you can declare a new variable, call it result and output some simple arithmetic:

```
int result;

result = a + b;
cout << result;
```

Insert the above into the code as per the screenshot.

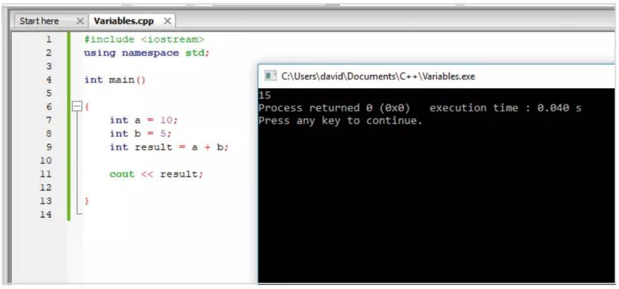


```
#include <iostream>
using namespace std;

int main()

{

    int a;
    int b;
    int result;

    a = 10;
    b = 5;
    result = a + b;

    cout << result;

}
```

```
15
Process returned 0 (0x0)   execution time : 0.045 s
Press any key to continue.
```

STEP 5 You can assign a value to a variable as soon as you declare it. The code you've typed in could look like this, instead:

```
int a = 10;
int b = 5;
int result = a + b;

cout << result;
```



```
#include <iostream>
using namespace std;

int main()

{
    int a = 10;
    int b = 5;
    int result = a + b;

    cout << result;

}
```

C:\Users\david\Documents\C++\Variables.exe

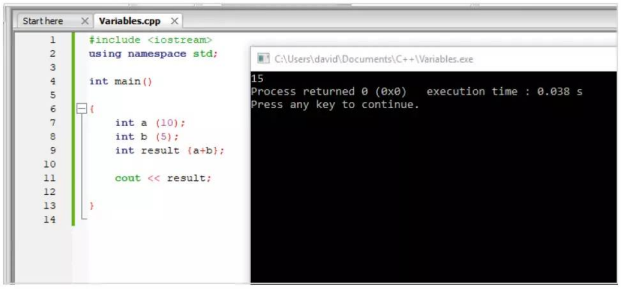
```
15
Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

STEP 6 Specific to C++, you can also use the following to assign values to a variable as soon as you declare them:

```
int a (10);
int b (5);
```

Then, from the C++ 2011 standard, using curly brackets:

```
int result {a+b};
```



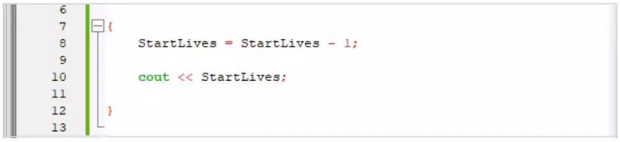
```
#include <iostream>
using namespace std;

int main()

{
    int a (10);
    int b (5);
    int result {a+b};

    cout << result;

}
```

C:\Users\david\Documents\C++\Variables.exe

```
15
Process returned 0 (0x0)   execution time : 0.038 s
Press any key to continue.
```

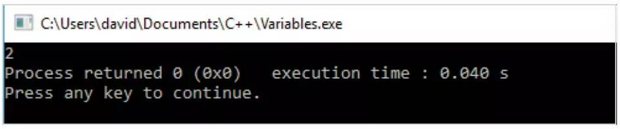
STEP 7 You can create global variables, which are variables that are declared outside any function and used in any function within the entire code. What you've used so far are local variables: variables used inside the function. For example:

```
#include <iostream>
using namespace std;
int StartLives = 3;

int main ()

{
    startLives = StartLives – 1;
    cout << StartLives;
}
```



```
{

    StartLives = StartLives - 1;

    cout << StartLives;

}
```

STEP 8 The previous step creates the variable StartLives, which is a global variable. In a game, for example, a player's lives go up or down depending on how well or how bad they're doing. When the player restarts the game, the StartLives returns to its default state: 3. Here we've assigned 3 lives, then subtracted 1, leaving 2 lives left.

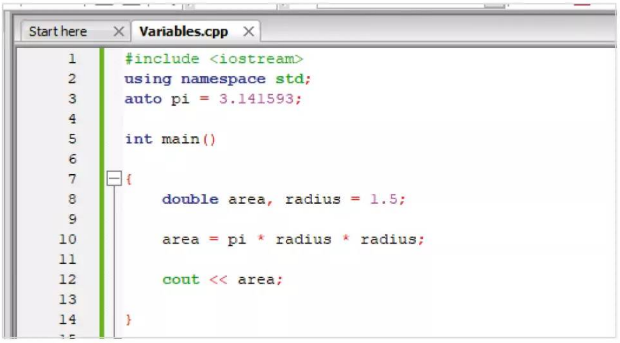


C:\Users\david\Documents\C++\Variables.exe

```
2
Process returned 0 (0x0)   execution time : 0.040 s
Press any key to continue.
```

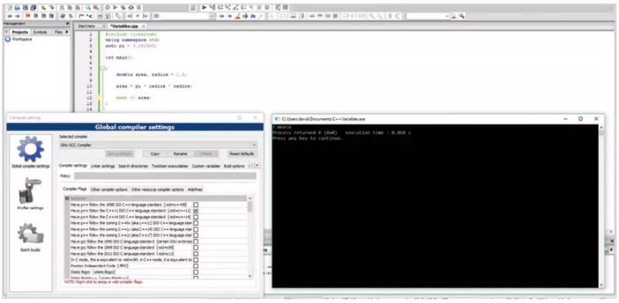
STEP 9 The modern C++ compiler is far more intelligent than most programmers give it credit. While there are numerous data types you can declare for variables, you can in fact use the auto feature:

```
#include <iostream>
using namespace std;
auto pi = 3.141593;

int main()

{
    double area, radius = 1.5;

    area = pi * radius * radius;

    cout << area;

}
```



```
#include <iostream>
using namespace std;
auto pi = 3.141593;

int main()

{
    double area, radius = 1.5;

    area = pi * radius * radius;

    cout << area;

}
```

STEP 10 A couple of new elements here: first, auto won't work unless you go to Settings > Compiler and tick the box labelled 'Have G++ follow the C++11 ISO C++ Language Standard [-std=c++1]'. Then, the new data type, double, which means double-precision floating point value. Enable C++11, then build and run the code. The result should be 7.06858.



Data Types

Variables, as we've seen, store information that the programmer can then later call up, and manipulate if required. Variables are simply reserved memory locations that store the values the programmer assigns, depending on the data type used.

THE VALUE OF DATA

There are many different data types available for the programmer in C++, such as an integer, floating point, Boolean, character and so on. It's widely accepted that there are seven basic data types, often called Primitive Built-in Types; however, you can create your own data types should the need ever arise within your code.

The seven basic data types are:

TYPE	COMMAND
Integer	`Integer`
Floating Point	`float`
Character	`char`
Boolean	`bool`
Double Floating Point	`double`
Wide Character	`wchar_t`
No Value	`void`

These basic types can also be extended using the following modifiers: Long, Short, Signed and Unsigned. Basically this means the modifiers can expand the minimum and maximum range values for each data type. For example, the int data type has a default value range of -2147483648 to 2147483647, a fair value, you would agree.

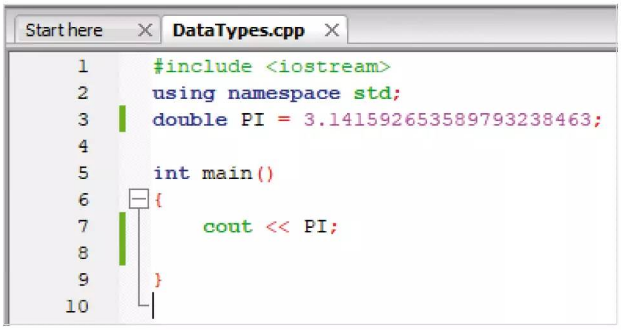
Now, if you were to use one of the modifiers, the range alters:

Unsigned int = 0 to 4294967295
Signed int = -2147483648 to 2147483647
Short int = -32768 to 32767
Unsigned Short int = 0 to 65,535
Signed Short int = -32768 to 32767
Long int = -2147483647 to 2147483647
Signed Long int = -2147483647 to 2147483647
Unsigned Long int = 0 to 4294967295

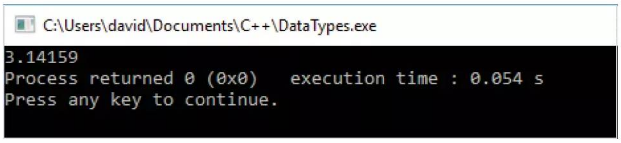
Naturally you can get away with using the basic type without the modifier, as there's plenty of range provided with each data type. However, it's considered good C++ programming practise to use the modifiers when possible.

There are issues when using the modifiers though. Double represents a double-floating point value, which you can use for incredibly accurate numbers but those numbers are only accurate up to the fifteenth decimal place. There's also the problem when displaying such numbers in C++ using the cout function, in that cout by default only outputs the first five decimal places. You can combat that by adding a cout.precision () function and adding a value inside the brackets, but even then you're still limited by the accuracy of the double data type. For example, try this code:

```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

int main()
{
    cout << PI;
}
```



```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

int main()
{
    cout << PI;

}
```

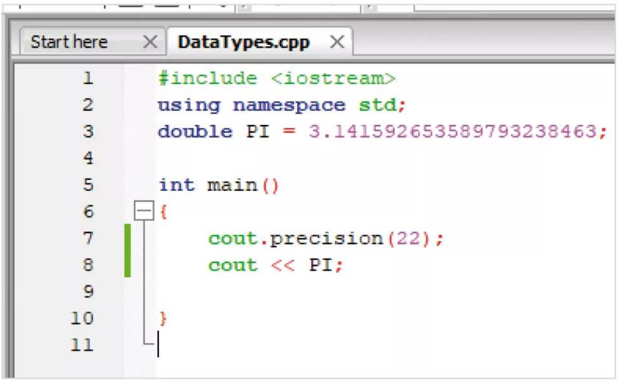


```
3.14159
Process returned 0 (0x0)   execution time : 0.054 s
Press any key to continue.
```

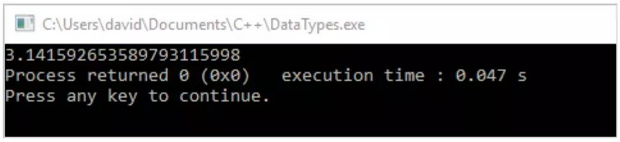
Build and run the code and as you can see the output is only 3.14159, representing cout's limitations in this example.

You can alter the code including the aforementioned cout.precision function, for greater accuracy. Take precision all the way up to 22 decimal places, with the following code:

```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

int main()
{
    cout.precision(22);
    cout << PI;
}
```

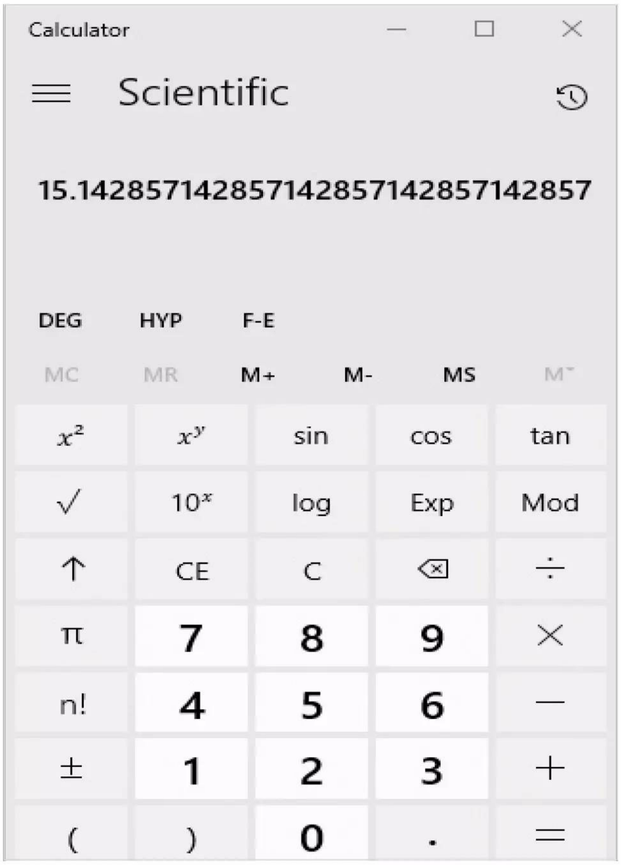

Start here
DataTypes.cpp

```
#include <iostream>
using namespace std;
double PI = 3.141592653589793238463;

int main()
{
    cout.precision(22);
    cout << PI;

}
```


C:\Users\david\Documents\C++\DataTypes.exe

```
3.141592653589793115998
Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

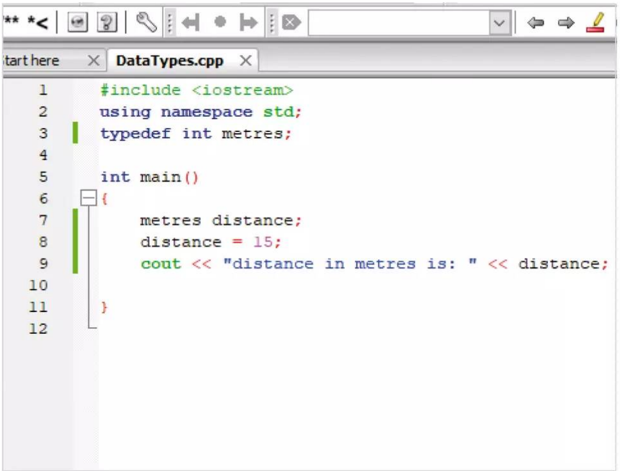
Again, build and run the code; as you can see from the command line window, the number represented by the variable PI is different to the number you've told C++ to use in the variable. The output reads the value of PI as 3.141592653589793115998, with the numbers going awry from the fifteenth decimal place.


Calculator
Scientific
15.142857142857142857142857142857
DEG
HYP
F-E
MC
MR
M+
M-
MS
M˅
x²
xʸ
sin
cos
tan
√
10ˣ
log
Exp
Mod
↑
CE
C
⌫
÷
π
7
8
9
×
n!
4
5
6
−
±
1
2
3
+
(
)
0
.
=

This is mainly due to the conversion from binary in the compiler and that the IEEE 754 double precision standard occupies 64-bits of data, of which 52-bits are dedicated to the significant (the significant digits in a floating-point number) and roughly 3.5-bits are taken holding the values 0 to 9. If you divide 53 by 3.5, then you arrive at 15.142857 recurring, which is 15-digits of precision.

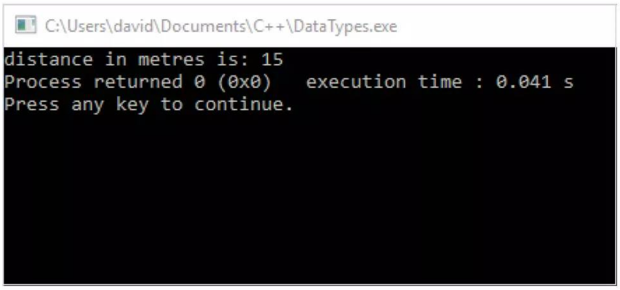
To be honest, if you're creating code that needs to be accurate to more than fifteen decimal places, then you wouldn't be using C++, you would use some scientific specific language with C++ as the connective tissue between the two languages.

You can create your own data types, using an alias-like system called typedef. For example:


tart here
DataTypes.cpp

```
#include <iostream>
using namespace std;
typedef int metres;

int main()
{
    metres distance;
    distance = 15;
    cout << "distance in metres is: " << distance;

}
```

```
#include <iostream>
using namespace std;
typedef int metres;

int main()
{
    metres distance;
    distance = 15;
    cout << “distance in metres is: “ << distance;

}
```


C:\Users\david\Documents\C++\DataTypes.exe

```
distance in metres is: 15
Process returned 0 (0x0)   execution time : 0.041 s
Press any key to continue.
```

This code when executed creates a new int data type called metres. Then, in the main code block, there's a new variable called distance, which is an integer; so you're basically telling the compiler that there's another name for int. We assigned the value 15 to distance and displayed the output: distance in metres is 15.

It might sound a little confusing to begin with but the more you use C++ and create your own code, the easier it becomes.

Strings

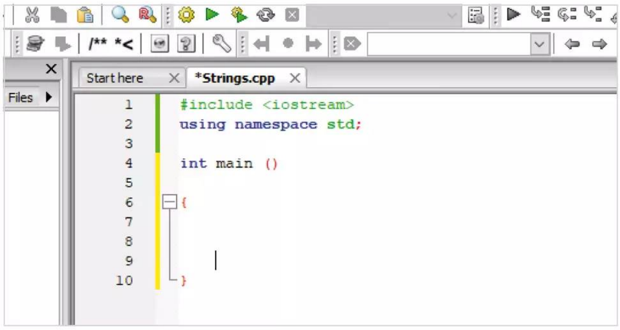
Strings are objects that represent and hold sequences of characters. For example, you could have a universal greeting in your code ‘Welcome’ and assign that as a string to be called up wherever you like in the program.

STRING THEORY

There are different ways in which you can create a string of characters, which historically are all carried over from the original C language, and are still supported by C++.

STEP 1 To create a string you use the char function. Open a new C++ file and begin with the usual header:

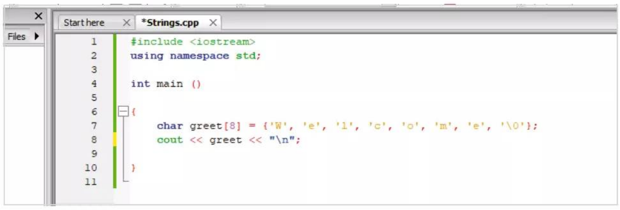
```
#include <iostream>
using namespace std;

int main ()

{

}
```



```
#include <iostream>
using namespace std;

int main ()

{


}
```

STEP 2 It’s easy to confuse a string with an array. Here’s an array, which can be terminated with a null character:

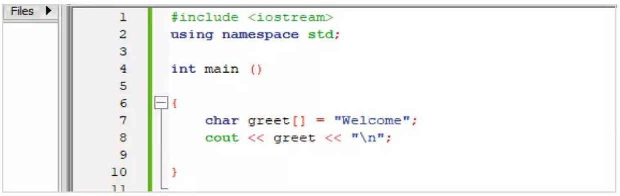
```
#include <iostream>
using namespace std;

int main ()

{
    char greet[8] = {‘W’, ‘e’, ‘l’, ‘c’, ‘o’, ‘m’,
‘e’, ‘\0’};
    cout << greet << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()

{
    char greet[8] = {'W', 'e', 'l', 'c', 'o', 'm', 'e', '\0'};
    cout << greet << "\n";

}
```

STEP 3 Build and run the code, and ‘Welcome’ appears on the screen. While this is perfectly fine, it’s not a string. A string is a class, which defines objects that can be represented as a stream of characters and doesn’t need to be terminated like an array. The code can therefore be represented as:

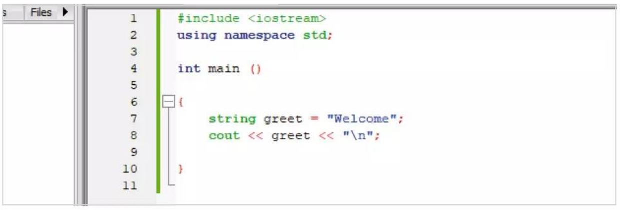
```
#include <iostream>
using namespace std;

int main ()

{
    char greet[] = “Welcome”;
    cout << greet << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()

{
    char greet[] = "Welcome";
    cout << greet << "\n";

}
```

STEP 4 In C++ there’s also a string function, which works in much the same way. Using the greeting code again, you can enter:

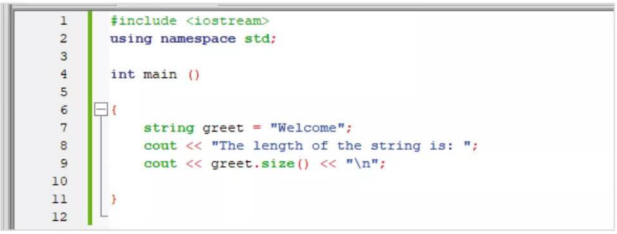
```
#include <iostream>
using namespace std;

int main ()

{
    string greet = “Welcome”;
    cout << greet << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()

{
    string greet = "Welcome";
    cout << greet << "\n";

}
```

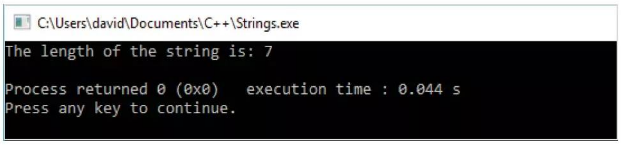
STEP 5 There are also many different operations that you can apply with the string function. For instance, to get the length of a string you can use:

```
#include <iostream>
using namespace std;

int main ()

{
    string greet = “Welcome”;
    cout << “The length of the string is: “;
    cout << greet.size() << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main  ()

{
    string greet = "Welcome";
    cout << "The length of the string is: ";
    cout << greet.size() << "\n";

}
```

STEP 6 You can see that we used greet.size() to output the length, the number of characters there are, of the contents of the string. Naturally, if you call your string something other than greet, then you need to change the command to reflect this. It's always stringname.operation. Build and run the code to see the results.

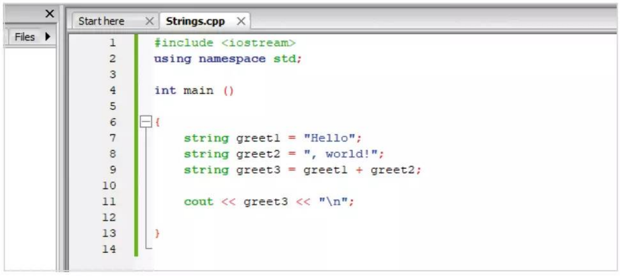


C:\Users\david\Documents\C++\Strings.exe

```
The length of the string is: 7

Process returned 0 (0x0)   execution time : 0.044 s
Press any key to continue.
```

STEP 7 You can of course add strings together, or rather combine them to form longer strings:

```
#include <iostream>
using namespace std;

int main ()

{
    string greet1 = “Hello”;
    string greet2 = “, world!”;
    string greet3 = greet1 + greet2;

    cout << greet3 << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main  ()

{
    string greet1 = "Hello";
    string greet2 = ", world!";
    string greet3 = greet1 + greet2;

    cout << greet3 << "\n";

}
```

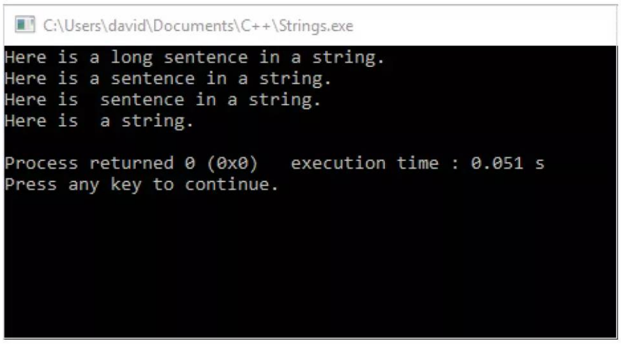
STEP 8 Just as you might expect, you can mix in an integer and store something to do with the string. In this example, we created int length, which stores the result of string.size() and outputs it to the user:

```
#include <iostream>
using namespace std;

int main ()

{
    int length;
    string greet1 = “Hello”;
    string greet2 = “, world!”;
    string greet3 = greet1 + greet2;

    length = greet3.size();

    cout << “The length of the combined strings
is: “ << length << “\n”;

}
```

STEP 9 Using the available operations that come with the string function, you can manipulate the contents of a string. For example, to remove characters from a string you could use:

```
#include <iostream>
using namespace std;

int main ()

{
  string strg (“Here is a long sentence in a
string.”);
  cout << strg << ‘\n’;

  strg.erase (10,5);
  cout << strg << ‘\n’;

  strg.erase (strg.begin()+8);
  cout << strg << ‘\n’;

  strg.erase (strg.begin()+9, strg.end()-9);
  cout << strg << ‘\n’;

}
```

STEP 10 It's worth spending some time playing around with the numbers, which are the character positions in the string. Occasionally, it can be hit and miss whether you get it right, so practice makes perfect. Take a look at the screenshot to see the result of the code.



C:\Users\david\Documents\C++\Strings.exe

```
Here is a long sentence in a string.
Here is a sentence in a string.
Here is  sentence in a string.
Here is  a string.

Process returned 0 (0x0)   execution time : 0.051 s
Press any key to continue.
```

C++ Maths

Programming is mathematical in nature and as you might expect, there's plenty of built-in scope for some quite intense maths. C++ has a lot to offer someone who's implementing mathematical models into their code. It can be extremely complex or relatively simple.

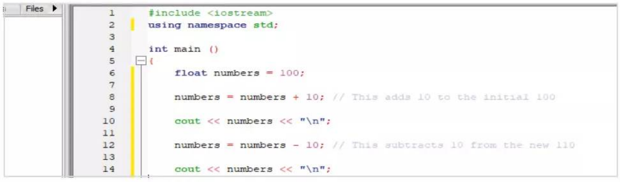
C++ = MC^2

The basic mathematical symbols apply in C++ as they do in most other programming languages. However, by using the C++ Math Library, you can also calculate square roots, powers, trig and more.

STEP 1

C++'s mathematical operations follow the same patterns as those taught in school, in that multiplication and division take precedence over addition and subtraction. You can alter that though. For now, create a new file and enter:

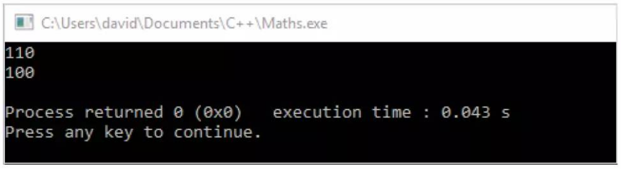
```
#include <iostream>
using namespace std;

int main ()
{
    float numbers = 100;

    numbers = numbers + 10; // This adds 10 to the
initial 100

    cout << numbers << “\n”;

    numbers = numbers - 10; // This subtracts 10
from the new 110

    cout << numbers << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{
    float numbers = 100;

    numbers = numbers + 10; // This adds 10 to the initial 100

    cout << numbers << "\n";

    numbers = numbers - 10; // This subtracts 10 from the new 110

    cout << numbers << "\n";
```

STEP 2

While simple, it does get the old maths muscle warmed up. Note that we used a float for the numbers variable. While you can happily use an integer, if you suddenly started to use decimals, you would need to change to a float or a double, depending on the accuracy needed. Run the code and see the results.

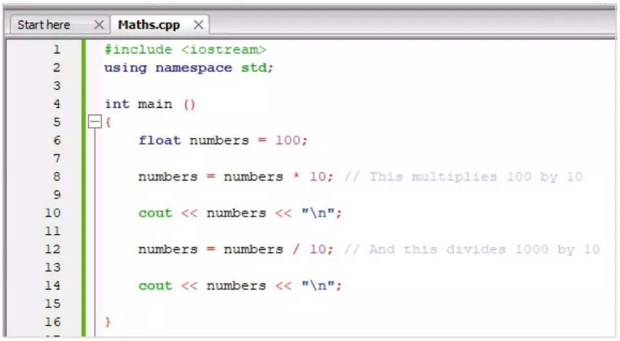


```
110
100

Process returned 0 (0x0)   execution time : 0.043 s
Press any key to continue.
```

STEP 3

Multiplication and division can be applied as such:

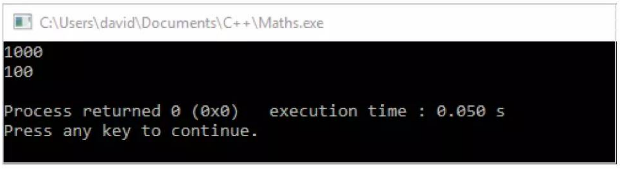
```
#include <iostream>
using namespace std;

int main ()
{
    float numbers = 100;

    numbers = numbers * 10; // This multiplies 100
by 10

    cout << numbers << “\n”;

    numbers = numbers / 10; // And this divides
1000 by 10

    cout << numbers << “\n”;

}
```



```
#include <iostream>
using namespace std;

int main ()
{
    float numbers = 100;

    numbers = numbers * 10; // This multiplies 100 by 10

    cout << numbers << "\n";

    numbers = numbers / 10; // And this divides 1000 by 10

    cout << numbers << "\n";

}
```

STEP 4

Again, execute the simple code and see the results. While not particularly interesting, it's a start into C++ maths. We used a float here, so you can play around with the code and multiply by decimal places, as well as divide, add and subtract.

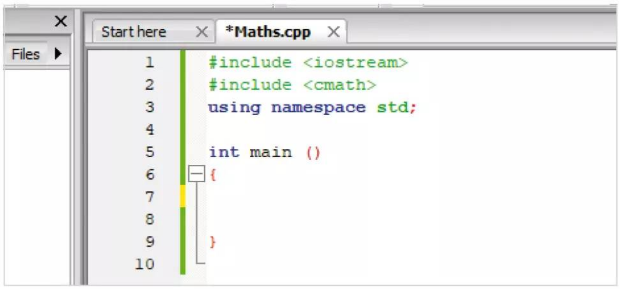


```
1000
100

Process returned 0 (0x0)   execution time : 0.050 s
Press any key to continue.
```

STEP 5

The interesting maths content comes when you call upon the C++ Math Library. Within this header are dozens of mathematical functions along with further operations. Everything from computing cosine to arc tangent with two parameters, to the value of PI. You can call the header with:

```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
}
```

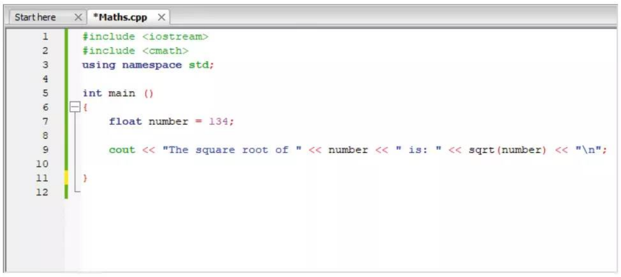


```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{


}
```

STEP 6

Start by getting the square root of a number:

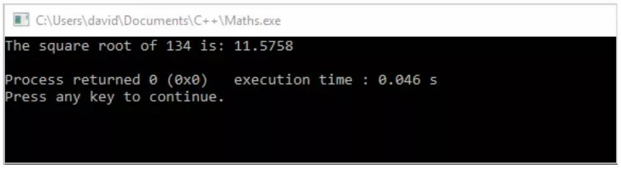
```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    float number = 134;

    cout << “The square root of “ << number << “
is: “ << sqrt(number) << “\n”;

}
```



```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    float number = 134;

    cout << "The square root of " << number << " is: " << sqrt(number) << "\n";
}
```

STEP 7

Here we created a new float called number and used the sqrt(number) function to display the square root of 134, the value of the variable, number. Build and run the code, and your answer reads 11.5758.

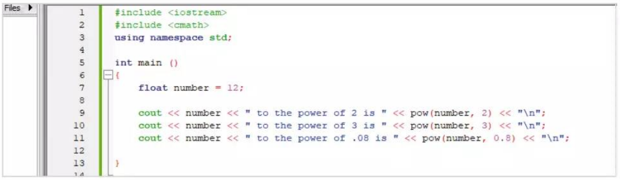


```
The square root of 134 is: 11.5758

Process returned 0 (0x0)   execution time : 0.046 s
Press any key to continue.
```

STEP 8

Calculating powers of numbers can be done with:

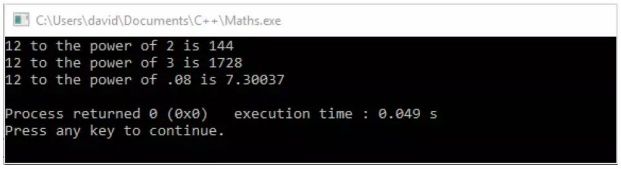
```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    float number = 12;

    cout << number << “ to the power of 2 is “ <<
pow(number, 2) << “\n”;
    cout << number << “ to the power of 3 is “ <<
pow(number, 3) << “\n”;
    cout << number << “ to the power of .08 is “
<< pow(number, 0.8) << “\n”;

}
```



```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    float number = 12;

    cout << number << " to the power of 2 is " << pow(number, 2) << "\n";
    cout << number << " to the power of 3 is " << pow(number, 3) << "\n";
    cout << number << " to the power of .08 is " << pow(number, 0.8) << "\n";

}
```

STEP 9

Here we created a float called number with the value of 12, and the pow(variable, power) is where the calculation happens. Of course, you can calculate powers and square roots without using variables. For example, pow (12, 2) outputs the same value as the first cout line in the code.

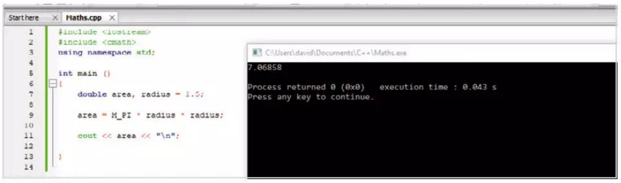


```
12 to the power of 2 is 144
12 to the power of 3 is 1728
12 to the power of .08 is 7.30037

Process returned 0 (0x0)   execution time : 0.049 s
Press any key to continue.
```

STEP 10

The value of Pi is also stored in the cmath header library. It can be called up with the M_PI function. Enter cout << M_PI; into the code and you get 3.14159; or you can use it to calculate:

```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    double area, radius = 1.5;

    area = M_PI * radius * radius;

    cout << area << “\n”;

}
```



```
#include <iostream>
#include <cmath>
using namespace std;

int main ()
{
    double area, radius = 1.5;

    area = M_PI * radius * radius;

    cout << area << "\n";

}
```

```
7.06858

Process returned 0 (0x0)   execution time : 0.043 s
Press any key to continue.
```

Save a whopping

25% Off!

ALL Tech Manuals

with Papercut

The iOS 17 iPhone Guidebook
The Complete Chromebook User Manual
The Windows 11 Manual
Coding For Beginners
The Samsung Galaxy Manual
The iPhone 15 Manual
The Complete Home Networking Manual
The Complete Build Your Own PC Manual
The Apple Watch Manual
The Complete C++ & Python Coding Manual

Not only can you learn new skills and master your tech, but you can now SAVE 25% off all of our coding and consumer tech digital and print guidebooks!

Simply use the following exclusive code at checkout:

NYHF23CN

www.pclpublications.com

C++ & Python For Beginners
22 - ISBN: 978-1-912847-12-9
Published by: Papercut Limited
Digital distribution by: ReadlyAB - www.readly.com

© 2025 Papercut Limited All rights reserved. No part of this publication may be reproduced in any form, stored in a retrieval system or integrated into any other publication, database or commercial programs without the express written permission of the publisher. Under no circumstances should this publication and its contents be resold, loaned out or used in any form by way of trade without the publisher's written permission. While we pride ourselves on the quality of the information we provide, Papercut Limited reserves the right not to be held responsible for any mistakes or inaccuracies found within the text of this publication. Due to the nature of the software industry, the publisher cannot guarantee that all apps and software will work on every version of device. It remains the purchaser's sole responsibility to determine the suitability of this book and its content for whatever purpose. Any app images reproduced on the front cover are solely for design purposes and are not representative of content.

We advise all potential buyers to check listing prior to purchase for confirmation of actual content. All editorial opinion herein is that of the reviewer - as an individual - and is not representative of the publisher or any of its affiliates. Therefore the publisher holds no responsibility in regard to editorial opinion and content.

This is an independent publication and as such does not necessarily reflect the views or opinions of the producers of apps or products contained within. This publication is 100% unofficial. All copyrights, trademarks and registered trademarks for the respective companies are acknowledged.

Relevant graphic imagery reproduced with courtesy of brands and products. Additional images contained within this publication are reproduced under licence from Shutterstock. Prices, international availability, ratings, titles and content are subject to change.

All information was correct at time of publication. Some content may have been previously published in other volumes or titles.

Papercut Limited
Registered in England & Wales No: 04308513

ADVERTISING – For our latest media packs please contact:
Brad Francis - brad@papercutltd.co.uk
Web - www.pclpublications.com

INTERNATIONAL LICENSING – Papercut Limited has many great publications and all are available for licensing worldwide.
For more information email: james@papercutltd.co.uk